THESE DE DOCTORAT CONJOINT TELECOM SUDPARIS et L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité : Informatique

Ecole doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

Wassim DRIRA

Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS

Un système de collecte sécurisé et de gestion des données pour les réseaux de capteurs sans fils

Soutenue le 10 décembre 2012

devant le jury composé de :

Rapporteurs	**Pr. Dijiang Huang**, *Arizona State University, USA*
	Pr. Pascal Berthomé, *ENSI-Bourges, France*
Examinateurs	**Pr. Guy Pujolle**, *Université Paris 6, France*
	Pr. Khaldoun Al Agha, *Université Paris-Sud, France*
	Pr. Jean-Louis Lanet, *Université de Limoges, France*
	Dr. Samia Bouzefrane, *CNAM, France*
Directeurs de thèse	**Pr. Djamal Zeghlache**, *Télécom SudParis, France*
	Dr. Éric Renault, *Télécom SudParis, France*

Thèse n° 2012TELE0051

Abstract

In the last decade, wireless sensor network (WSN) domain had benefit from a huge development effort and a major technological boom of Micro-Electro-Mechanical Systems which make, nowadays, each user or organization already connected to a large number of nodes (mobile phone, network monitoring, sensors in the home, the sensors on the body, etc. .). These nodes generate a substantial amount of data, making the management and storage of data not an obvious issue. However, these nodes have, in general, a limited memory and processing capabilities, so they are unable to store and manage the associated data flow. In addition, these data can contain user's confidential data (location, health, etc.). For these reasons, developing a secure system managing the collection, storage, indexing, sharing of data and alerts generation from heterogeneous sensor nodes is a real need for users and organizations.

In the first part of the thesis, we developed a *middleware* for wireless sensor networks to communicate with the physical sensors for storing, processing, indexing, analyzing and generating alerts on those sensors data. The *middleware* is composite-based system. A composite is a software component that is connected to a physical node like a sensor node, a mobile phone or a gateway, etc. or used to aggregate and process data from different composites. Each physical node that has the capability to communicate with the middleware should be setup as a composite. A composite is a set of instances of components interconnected using services. There are some default components while new components can be added easily. The *middleware* has been tested and used in the context of the European project Mobesens in order to receive, store, process, index and analyze data from a sensor network for monitoring water quality.

In the second part of the thesis, we proposed a new hybrid authentication and key establishment scheme that will focus on the relationship between the three parties forming Wireless Body Area Networks (WBANs), e.g. the senor node (SN), the mobile node (MN) and the storage server (SS) or the middleware. The scheme combines symmetric cryptography and identity-based cryptography. Nodes having scarce resources use symmetric keys, while those having more resources use asymmetric keys. It is based on two protocols. The first protocol intent is the mutual authentication between SS and MN, on providing an asymmetric pair of keys for MN, and on establishing a pairwise key between them. The second protocol aims at authenticating them, and establishing a group key and pairwise keys between SN and the two others.

The *middleware* that was originally designed to be used by a single user or organization, has been generalized in the third part of the thesis in order to provide a private space for each organization or user to manage his sensors data using *cloud computing*. Next, we expanded the composite with gadgets that can be integrated into the portal of the organization, the user or a third party portal to share sensor data and then provide a social portal for sensor networks.

Acknowledgements

It is a pleasure to thank all those people who made this thesis possible.

I would like to express my deep gratitude to my supervisor, *Prof. Djamal Zeghlache*, for welcoming me in his research group at Télécom SudParis and for providing me with the support and guidance that made this work possible. I also would like to express my deep gratitude to my advisor *Dr. Éric Renault* for the continuous support during my Ph.D study and research, for his patience, motivation and enthusiasm. He was always available to advise me in all time both for research and everyday life.

Besides my advisors, I would like to thank the rest of my dissertation committee members, Dr. Samia Bouzefrane, Prof. Pascal Berthomé, Prof. Dijiang Huang, Prof. Guy Pujolle, Prof. Khaldoun Al Agha and Prof. Jean-Louis Lanet for their precious time and valuable suggestions for the work done in this dissertation. Special thanks goes to Prof. Pascal Berthomé and Prof. Dijiang Huang for their effort in preparing their reports in record time in order to allow me to defend in time.

I extend heartfelt thanks to my friends and my colleagues at Télécom SudParis for their support over the years. In particular, I thank Houssem Medhioub, Mohamed Abid and Aymen Boudguiga for their generous friendship and their wise counsel. Moreover, a special acknowledgement is necessary for the administrative staff and especially the department secretary, Valérie Mateus, for their continuous effort to facilitate administrative procedures.

I would like to express my appreciation to all my teachers for their support and high quality courses.

This Ph.D. project would not have been possible without the financial support of the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 223975, project "Mobesens".

Lastly, and most importantly, my special and deepest appreciations go out to my family members to whom I owe so much. I thank my parents, Mohamed Drira and Fathia Béjar. They bore me, raised me, supported me, taught me, and loved me. I would like to thank my brothers. My appreciation also goes to my father- and mother-in-law for their encouragement. Finally, I would like to thank my dearest wife Mariem, my beautiful girl Emna and our future baby. They make my life beautiful and they gave me encouragements, support and patience to achieve this thesis.

Contents

List of Figures

List of Tables

Acronyms

6LoWPAN IPv6 over Low-power Wireless Personal Area Networks. 15, 17, 18

API Application Programming Interface. 38, 51

B-DH Bilinear Diffie-Hellman. 64

BSN Body Sensor Network. 27

C-DH Computational Diffie-Hellman. 64, 88

FIFO First In First Out. 14

HTTP HyperText Transfer Protocol. 17, 32, 51

IBC Identity-Based Cryptography. 88

IoT Internet of Things. 15

IPv6 Internet Protocol version 6. 16, 17

ISFET Ion-Sensitive Field-Effect Transistor. 55, 91

JMS Java Message Service. 51, 53

MTU maximum transmission unit. 16

OS Operating System. 2, 4, 13, 14, 26, 32, 87

PAN Personal Area Network. 17

QoS Quality of Service. 10, 11, 15

RA Resource Allocator. 76, 77

RAM Random Access Memory. 8

REST REpresentational State Transfer. 17, 18

SFTP Secure File Transfet Protocol. 51

SN Sensor Node. 4, 5, 27, 38, 87–89

SNS Social Network Site. 29

SWE Sensor Web Enablement. 21, 88

UID Unique IDentifier. 52, 53

URI Uniform Resource Identifier. 17

VM Virtual Machine. 26, 27, 76, 77, 89

VSN Virtual Sensor Network. 26, 28, 29

WAN Wide Area Network. 15

WBAN Wireless Body Area Network. 88

WPAN Wireless Personal Area Network. 15

WSN Wireless Sensor Network. 1–4, 7, 9–15, 17–20, 26, 28–32, 38, 51, 87, 88

WWW World Wide Web. 17

Résumé

Les progrès des technologies micro-électromécaniques, électroniques, numériques et des communications sans fil ont permis le développement de capteurs (SN) peu coûteux communiquant à faible distance et peuvant traiter et stocker des données. Ces nœuds minuscules ont la capacité de surveiller un ou plusieurs phénomènes physiques et de s'auto-configurer entre eux dès leurs démarrage afin de former un réseau de capteurs sans fil (RCSF).

Le développement à grande échelle de ces réseaux, met à disposition de chaque utilisateur ou organisation un nombre important de nœuds (téléphones mobiles, réseaux de surveillance, capteurs à la maison, capteurs sur le corps, etc.). Ces nœuds génèrent une quantité importante de données, faisant ainsi de la gestion et du stockage de ces données une tâche difficile. Ces nœuds sont généralement caractérisés par une faible capacité de stockage et de traitement ; ils sont donc incapables de stocker et de gérer les flux de données associés. De plus, ces données peuvent contenir des informations concernant la vie privée de l'utilisateur (localisation, santé, etc.). Il en découle que le développement d'un système sécurisé de collecte, de gestion, de stockage et de partage de données collectées par les capteurs est un véritable besoin pour les utilisateurs et les organisations.

1 La gestion des données dans les RCSF

Le succès des réseaux sans fils, dont les réseaux de capteurs sans fils, est dû à l'utilisation de petits composants sans fil légers, autonomes et efficaces proposant divers services. La taille minuscule de ces nœuds constitue un avantage, mais présente aussi des contraintes en terme d'énergie, de bande passante, de mémoire et de capacité de calcul. Cela affecte l'ensemble du réseau et incite donc les chercheurs et les industriels à proposer des solutions matérielles, logicielles, ou architecturales afin d'assurer l'efficacité et la fiabilité de ces réseaux.

Ainsi, des applications spécifiques ont été élaborées pour s'exécuter sur des systèmes d'exploitation légers conçus spécialement pour ces capteurs. De plus, grâce à l'apparition

de nouvelles tendances en matière de programmation, des techniques telles que la virtualisation et le *Cloud Computing* ont été adoptées pour fournir des réseaux de capteurs plus flexibles, reconfigurables et performants. En effet, la performance d'un RCSF est mesurée entre autre par sa capacité de gérer la grande quantité de données générées par ces nœuds peuvant être homogènes ou hétérogènes. La collecte, l'analyse et le stockage des données des capteurs devient une tâche fastidieuse d'autant plus que le nombre de capteurs déployés dans le réseau peut être important. En outre, la faible capacité de stockage des nœuds complique cette gestion des données au sein du RCSF.

En raison de leur nature spécifique, les réseaux sans fil sont plus vulnérables aux agressions extérieures et présentent plus de failles de sécurité par rapport aux réseaux câblés. La sécurité dans les RCSF est également une préoccupation majeure qui ne doit pas être négligée. En fait, les nœuds sont généralement déployés sans protection ni surveillance. Par conséquent, ils rencontrent de nombreuses menaces physiques telles que la manipulation du nœud, l'injection de nœuds malicieux ou leur réplication.

La présence de capteurs dans plusieurs appareils mobiles ainsi que l'évolution et le déploiement des réseaux de capteurs sans fil dans plusieurs domaines font que chaque personne ou organisation est aujourd'hui en possession de beaucoup de capteurs, souvent hétérogènes. Construits sur différentes configurations et modes de fonctionnement, ces capteurs ne sont pas censés communiquer les uns avec les autres et génèrent de grande quantités de données avec des formats différents. La capacité à gérer et traiter ces données pour en extraire les informations utiles, ainsi que générer les alertes constitue un réel besoin confronté à de nombreuses limites et problèmes.

Ces limites sont à la fois matérielles et logicielles. Les limites matérielles sont dues aux ressources disponibles restreintes sur chaque capteur (traitement, communication, stockage, énergie), tandis que les limitations logicielles sont dues au protocole MAC utilisé dans les réseaux de capteurs permettant aux nœuds d'avoir des intervalles périodiques de sommeil afin de réduire leur consommation d'énergie, ceci rendant les nœuds périodiquement inaccessibles. Pour surmonter ce problème, l'utilisation d'un système externe est également envisageable. En outre, un compromis entre les deux approches de stockage dans le réseau et dans un système externe peut être réalisé. Néanmoins, lorsque les données doivent être conservées pendant une longue période, l'utilisation d'un système externe devient une obligation, ce qui améliore la qualité du réseau mais rajoute aussi d'autres problématiques à gérer.

Lorsqu'un utilisateur ou une application souhaite collecter des données, il envoie au réseau des requêtes ou des routines pour récupérer les mises à jour périodiques ou les alertes soulevées. Étant donné que ces derniers peuvent provenir de nœuds hétérogènes, le système de collecte utilisé dans le RCSF doit tenir compte de cette hétérogénéité et doit comprendre toutes les données afin d'être en mesure de bien les traiter. En effet, le traitement des données dans le réseau peut signifier l'exécution de certaines opérations

telles que l'analyse, l'agrégation, la conversion ou la traduction des données.

Le réseau de capteurs peut gérer des données publiques tout comme il peut être en charge de données très sensibles et privées. Or, les réseaux sans fil sont très vulnérables et peuvent être facilement manipuler par un utilisateur malveillant. Un choix multiple de protocoles de sécurité existe actuellement pour les réseaux traditionnels mais ils ne sont pas adaptés aux spécifités des RCSF. Un solution de sécurité sophistiquée et bien adaptée à ce type de réseaux et à ses exigences est donc impérative afin de préserver la vie privée des utilisateurs.

2 iSensors : un *middleware* dynamique et extensible

Afin de résoudre la problématique de collecte et de gestion des données hétérogènes des RCSF, cette section présente un *middleware* basé sur les notions de composants et d'événements.

2.1 Architecture

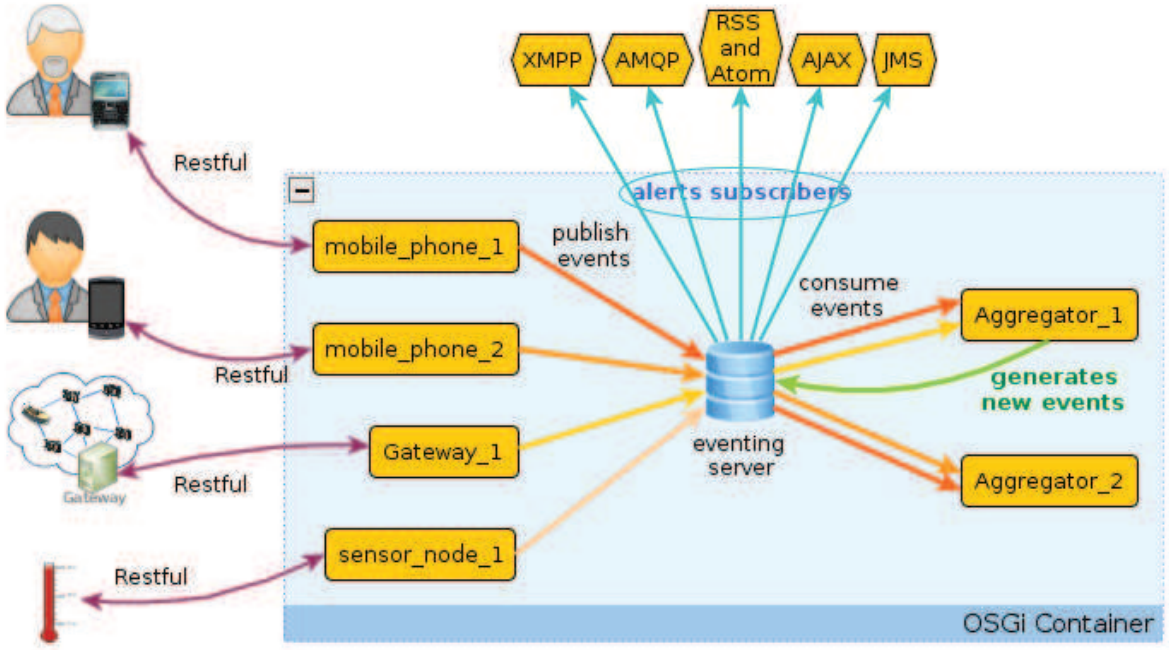


FIGURE 1 – Architecture globale du *middleware* iSensors.

Ce *middleware*, illustré en Fig. 1, est basé sur la notion de composants et de composites, dans lequel chaque nœud physique (passerelle, capteur, etc.) communique avec un composite du *middleware* qui lui ait dédié une interface RESTful. En plus des composites liés à des nœuds physiques, un autre type de composite est présent afin d'écouter

et d'agréger les alertes générées par d'autres composites. Pour former un composite, des composants par défaut sont offerts par le *middleware* pour être instancié et interconnecté. De nouveaux composants peuvent aussi être ajoutés à ce *middleware*. L'interconnexion, l'ajout et le changement de composants peut se faire à chaud, le *middleware* ayant l'intérêt d'être dynamique.

En plus des composites, un serveur d'événements est présent dans le *middleware* pour gérer la réception et la distribution des alertes. Il est basé sur le modèle Publish-Subscribe qui est un mécanisme de publication/souscription de messages dans lequel les émetteurs envoient leurs messages à des centres d'intérêt au lieu de les envoyer à des destinations spécifiques. L'abonné à un centre d'intérêt reçoit les publications des émetteurs. En effet, le serveur peut acheminer les événements du *middleware* pour les applications extérieures souscrites en utilisant de nombreux protocoles (RSS, JMS, XMPP, ajax, etc.). De plus, les composites peuvent être aussi bien des publicateurs et/ou des abonnés. Grâce à cette capacité de gestion des alertes, le serveur d'événements peut être considéré à la fois comme un mécanisme pour router les événements vers des applications externes souscrites, ainsi qu'un mécanisme de communication entre des composites hétérogènes. Un composite agrégateur se souscrit alors à des sujets d'intérêt afin d'agréger les données publiées.

2.2 La structure des composites

Afin de garantir une flexibilité et une reconfiguration facile des fonctionalités du *middleware*, la notion de composition a été sélectionnée. En effet, cette caractéristique permet de produire des composants indépendants et de les relier ensuite entre eux pour former un composite ou un programme plus sophistiqué.

Ainsi, dans ce *middleware*, chaque nœud est un composite qui instancie des composants et/ou d'autres composites. Par conséquent, chaque nœud forme un espace privée qui n'interfère pas avec les autres nœuds. Les composants sont reliés entre eux à l'aide de services. Chaque composant peut fournir ou consommer un service pour s'interconnecter aux autres composants. Sa configuration se fait à l'aide de l'un des services qu'il fournit, ou en utilisant le constructeur lors de l'instanciation. Certaines configurations peuvent également être faites en utilisant l'API RESTful pour ajouter ou modifier des *scripts* et des variables de configuration.

Afin de faciliter l'ajout de nouveaux nœuds au *middleware*, des composants produisant des services communs sont fournis par défaut. Ainsi, l'ajout d'un nouveau composite est limité à l'instanciation de certains composants par défaut. La Fig. 2 décrit la structure d'un composite représentant un nœud capteur ou une passerelle. Les composants les plus important qui y sont définis sont :

– **Le composant de stockage et d'indexation** est chargé du stockage permanent et de l'indexation des données. Il fournit les services d'enregistrement (*Put*), de

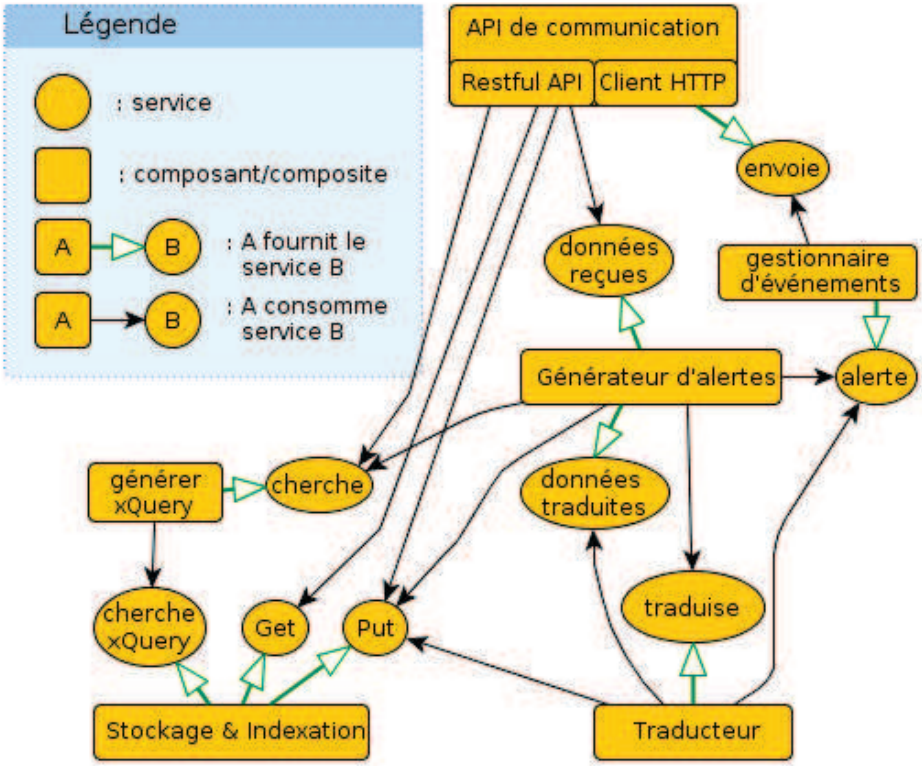


FIGURE 2 – Structure d'un composite de capteur ou passerelle.

récupération (*Get*) et de recherche des données. L'appel du service *Put* retourne un identifiant unique qui pourra être utilisé pour récupérer la donnée à l'aide du service *Get*. Un composant embarqué dans le *middleware* est fourni par défaut pour se connecter à la base de données non-SQL nommé existDB [1]. Ce service peut également être fourni par un autre composant afin de s'interfacer à une autre solution d'indexation ou de stockage, par exemple, en nuage.

- **Le générateur de requêtes xQuery** fournit une couche d'adaptation entre le *composant de stockage et d'indexation* et les autres composants utilisant le service de recherche. Ce composant traduit une requête générique en xQuery pour permettre aux consommateurs de son service d'être indépendants de la solution utilisée pour le stockage et l'indexation des données.
- **Le composant générateur d'alertes** est responsable de la génération des alertes après l'analyse des données reçues et/ou traduites. La génération est effectuée par l'exécution d'une fonction spécifique dans un script xQuery sur les données en entrée. Afin de permettre au script d'exécuter les opérations de raisonnement, la dernière version des données est injectée dans le *script*. À cette étape, le composant génère un message d'alerte et des méta-données supplémentaires peuvent être insérées aux données en entrée.
- **Le gestionnaire d'événements** est responsable du routage des événements reçus

par son service *alerte* au meilleur serveur disponible adapté à la gestion de ce type d'événements. Par exemple, les alertes XMPP [2] et JMS [3] sont redirigées vers le *serveur d'événements*, tandis qu'un autre serveur est nécessaire pour acheminer les SMS ou les e-mails.

- **Le composant de l'API de communication** implémente une API RESTful pour configurer et interagir avec les composites. L'API RESTful a été choisie car il s'agit d'une interface de services simple, sans état et compatible avec le standard HTTP. Tout d'abord, lors d'une communication, le composant reçoit des données des nœuds physiques. Il fournit alors un service pour envoyer des messages (réponse, étalonnage, re-configuration, etc.) pour le nœud physique. Ensuite, il joue le rôle d'interface pour naviguer et chercher dans les données des nœuds physiques. Enfin, il donne la possibilité de configurer les autres composants du composite, comme la mise à jour des fonctions xQuery du *générateur d'alertes*.
- **Le composant traducteur** est utilisé pour traduire et/ou ajouter des méta-données aux données reçues à l'aide d'une fonction xQuery. Par exemple, il peut être utilisé pour convertir les données de capteur reçues dans un format binaire en XML ou tout autre format de données. De plus, il peut ajouter un état de validation aux données ou tout autre méta-donnée.
- **Le composant agrégateur** est un élément du composite *agrégateur* utilisé pour exécuter un script sur plusieurs données en entrée afin de les agréger en se basant sur des paramètres de configuration et la fonction d'agrégation définie dans le script.
- **Le composant d'écoute** est un élément du composite *agrégateur* utilisé pour se souscrire aux sujets qui intéressent l'agrégateur afin de recevoir les nouveaux événements publiés et qui sont, ensuite, transférés au *composant agrégateur*.

2.3 Implémentation

Afin de garantir le concept de composition et la séparation des instances, les technologies OSGi et iPOJO ont été choisies pour implémenter le *middleware*. En effet, la technologie OSGi rassemble un ensemble de spécifications qui définissent un système de composants dynamiques. Ce framework offre un système modulaire et une plateforme de services permettant d'installer, démarrer, mettre à jour et ré-installer à distance des applications ou des composants sans pour autant re-démarrer le système.

De l'autre coté, la technologie iPOJO, une extension de la technologie OSGi, est un composant de services d'exécution visant à simplifier le développement d'applications OSGi. Il permet aux développeurs de séparer clairement le code fonctionnel (c.à.d. les POJO) du non-fonctionnel (gestion des dépendances, configuration, etc.) et c'est au moment de l'exécutiion que ces deux aspects sont combinés.

Ainsi, grâce au choix de ces technologies, le *middleware* est entièrement personnalis-

able. De nouveaux composants et services peuvent être facilement ajoutés à chaud dans la plateforme (sans redémarrage).

2.4 Extensions

Considérant que le *middleware* est basé sur des composants et des services, l'ajout d'extensions est à priori facile. Cette flexibilité ouvre donc les portes vers l'ajout de nouvelles fonctionnalités et services au *middleware*.

Une des extensions possibles est la sous-traitance des calculs. En effet, les réseaux de capteurs sont munis de ressources limitées alors que le *middleware*, installé sur un serveur à haute performance, possède plus de capacité de calcul. Dans ces circonstances, une bonne solution pour pallier les limites des capteurs est de permettre à l'utilisateur d'ajouter un composant/composite au *middleware* pour fournir ce service au capteur. En appelant ce service, le capteurs envoie les entrées et reçoit en sortie le résultat de l'exécution du service.

L'extension du *middleware* aux réseaux sociaux est bénéfique aux utilisateurs vu qu'elle permet le partage des données collectées par les capteurs. Afin de permettre ce partage entre un patient et son médecin ou entre diverses équipes de recherche peuvant collaborer au traitement et à l'analyse de ces données pour tirer des conclusions, le composite peut être étendu avec un gadget utilisé comme un *mashup web* à intégrer dans un portail social, voir Sec. 5.

3 Un système de stockage et de visualisation de données d'un RCSF

Le *middleware iSensors* décrit auparavant est extensible, flexible et peut servir dans divers domaines. Dans le cadre de cette thèse, nous avons choisi de l'utiliser et de le tester dans le cadre du projet Européen Mobesens [4] dans le but de recevoir, stocker, transformer, indexer et analyser les données d'un réseau de capteurs pour la surveillance de la qualité de l'eau. Un système de visualisation a aussi été intégré pour visualiser les données sur des clients légers (navigateurs web) en temps-réel.

3.1 Le projet mobesens

Le projet Mobesens [4] vise à développer des capteurs pour mesurer différents paramètres physico-chimiques de l'eau afin d'en apprécier sa qualité dans l'environnement. Ainsi, il met en place à la fois un réseau de communication et une infrastructure de typegrappe de calcul pour permettre aux données d'être transmises à partir des capteurs et ensuite stockées, traitées et affichées dans une interface utilisateur pratique et facile à utiliser. Il

est à noter que, même si le projet se limite à mesurer et analyser les paramètres de la qualité de l'eau, les travaux présentés peuvent être adaptés à tout autre type d'environnement, comme le sol, l'air, etc.

Dans ce projet, plusieurs réseaux de capteurs sans fil ont été déployés dans différents endroits : le lac de Genève, Brest, l'étang de Thau et au bord de l'Èbre. Chaque réseau est composé de nœuds de capteurs CSEM [5] implémentant les protocoles WiseMAC [6] et Wisenet [7]. Chaque noeud est relié à un/plusieurs capteurs physico-chimiques par l'intermédiaire d'une interface RS485. Les nœuds peuvent être mobiles ou statiques.

Afin de permettre la collecte et le transfert des données, une passerelle avec des capacités de communication plus large est placée à proximité du réseau de capteurs sans fil. Cette passerelle peut être soit fixe soit mobile, située en bord de mer ou dans l'eau. Ensuite, les données sont transférées au système de stockage et d'indexation pour permettre à la fois leur stockage permanent et d'effectuer des traitements plus complexes. Les données stockées dans le système sont disponibles via l'Internet. De plus, une interface de visualisation est fournie pour aider les utilisateurs finaux à visualiser les données afin de les surveiller, les analyser, etc. Cette interface Web a été développée pour permettre à divers groupes de recherche à travailler ensemble. En plus, elle est compatible avec une grande variété d'appareils comme les ordinateurs, les tablettes, les smartphones, etc. car elle a été développée en utilisant des technologies web. Dans ce travail, nous nous intéressons au système de gestion et de visualisation des données.

3.2 Architecture du système de gestion et de visualisation des données

En se basant sur la spécification du projet, le système de gestion et de visualisation des données doit fournir les fonctionalités suivantes :

- stocker les données de tous les éléments impliqués dans l'architecture, c'est-à-dire les mesures issues des capteurs, ainsi que les données utilisées pour leur gestion, ou toute autre donnée qui pourrait être utilisées ou échangées par différentes parties, par exemple, les informations de localisation des passerelles.
- permettre la visualisation de toutes les données des capteurs en temps-réel par le biais d'une application basée sur le Web.
- fournir une application web conviviale pour générer des graphiques, naviguer et valider les données de l'historique.
- visualiser les nœuds capteurs comme des points d'ancrage sur une carte pour faciliter leur localisation et leur suivi.

Dans le but de répondre à ces exigences, l'architecture du système, décrite en Fig. 3, est caractérisée par :

∗ **Le *middleware iSensors* :** la différence d'avec la version standard est que dans

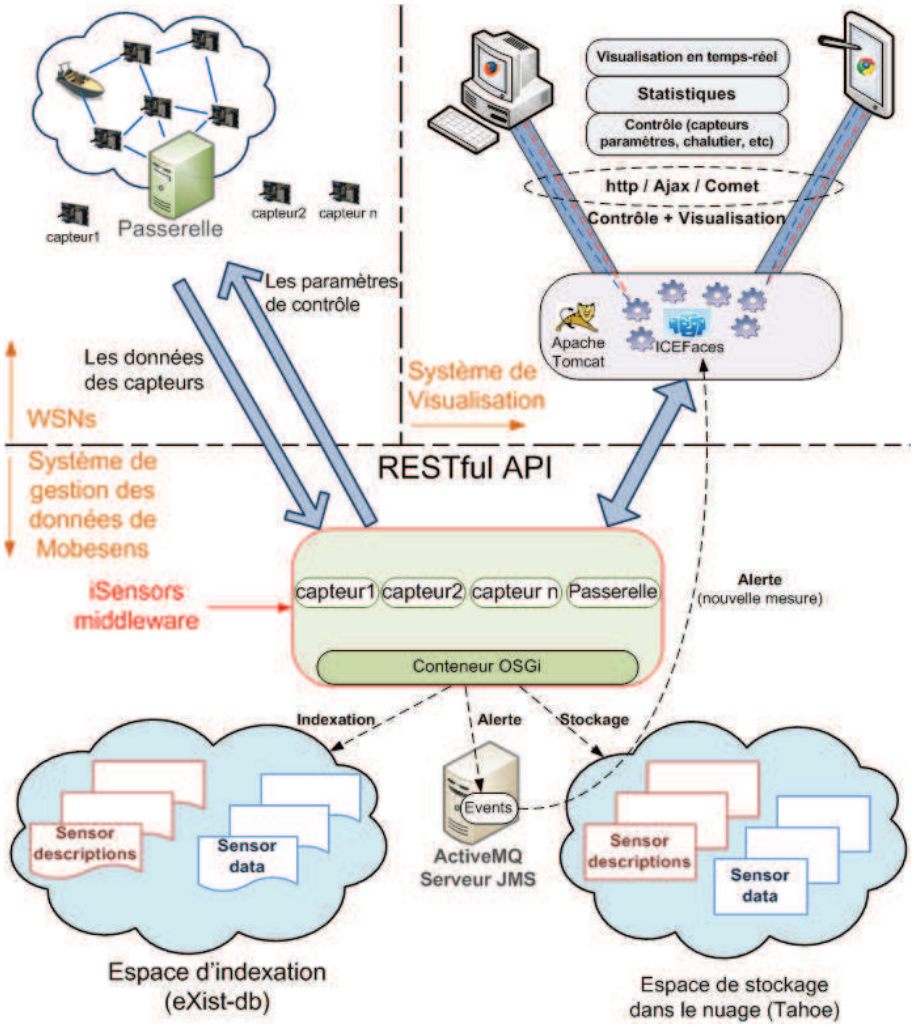


FIGURE 3 – Vu abstraite de l'architecture et implémentation du système.

celle ci, deux composants différents sont dépoloyés pour la gestion du stockage et l'indexations respectivement.

* **Un serveur d'événements :** ce serveur est l'un des éléments clés du système puisqu'il permet de fournir les mesures en temps-réel aux utilisateurs finaux. Cette partie a été réalisée en utilisant un serveur Apache ActiveMQ autonome.
* **Un espace de stockage :** cet espace vise à stocker les informations. Il est utilisé pour stocker les données brutes et les fichiers XML. Cet espace est basé sur Tahoe [8], une solution sécurisée de stockage dans le nuage.
* **Un espace d'indexation** qui compile toutes les métadonnées fournies par les nœuds capteurs, les utilisateurs finaux et les applications afin de permettre des recherches efficaces. En outre, il indexe les représentations XML des mesures. L'outil utilisé pour effectuer l'indexation est eXist-db [1], qui a l'avantage d'offrir une solution fiable et efficace pour l'indexation et la recherche de documents XML.
* **Une API RESTful :** elle permet aux capteurs de pousser leurs données collectées

dans le système et les clients Web pour visualiser ces données en utilisant la même API. Chaque composite inclut un composant de communication, tel que décrit dans la Fig. 2, qui fournit cette API.

* **Un système de visualisation :** ce système est responsable de l'affichage des données en temps-réel et la navigation dans les données composant l'historique. La page d'accueil de l'interface de visualisation contient une carte au dessus et dessous un tableau de données pour chaque type de capteurs dans un onglet dédié. Les nœuds capteurs sont représentés par des points d'ancrage cliquables sur la carte. Lors d'un clic, une fenêtre, affichant des informations sur le nœud et ses données récentes, est affichée. Le tableau des données affiche les données des capteurs en temps réel en ordre décroissant en foction de la date de réception.

3.3 Implémentation et tests

Le système a été implémenté et hébergé sur un serveur à Évry, France. Des campagnes de test et de déploiment ont été réalisées durant le projet devant la commission Européenne pour valider le système et le projet Mobesens. En parallèle, nous avons réalisé nos propres

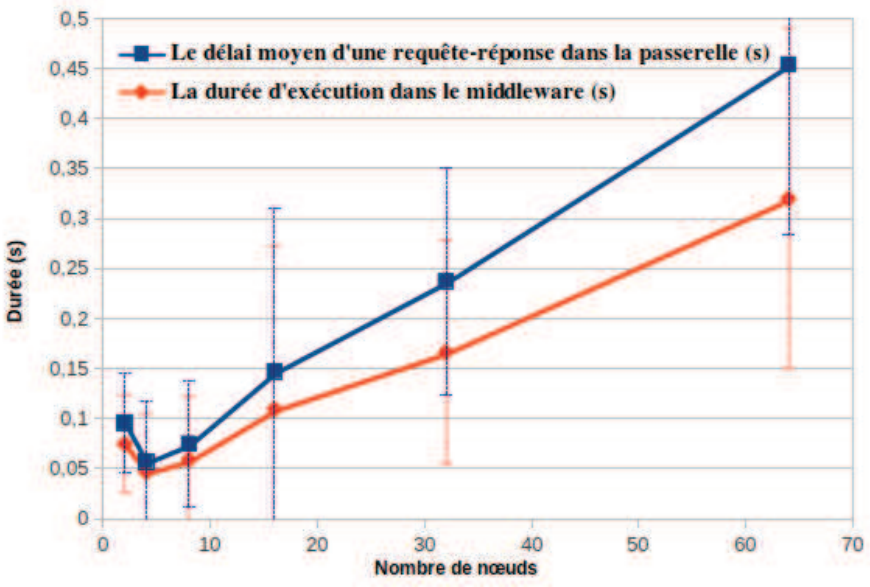


FIGURE 4 – Le délai de réponse du système en fonction du nombre de nœuds.

tests en laboratoire, parmi lesquels, un test pour émuler plusieurs nœuds qui envoient leurs données en même temps au *middleware*. La passerelle lance plusieurs processus en même temps pour envoyer un message d'une longueur de 40 octets au *middleware*. Les processus émulent les nœuds qui sont en train d'envoyer un message au *middleware*. Cette expérience est réalisée une centaine de fois pour chaque nombre de *threads*. La Fig. 4 affiche deux courbes : la bleue montre la moyenne et l'écart-type du délai entre l'émission du message

et la réception de la réponse du *middleware* pour chaque processus, et la rouge montre la moyenne et l'écart-type du temps d'exécution dans le système. À titre d'exemple, lorsque 64 nœuds démarrent leur communication avec le système en même temps, la courbe bleue montre que chaque communication se termine après une moyenne de 450 ms. Tandis que, le temps d'exécution pour les fonctions de stockage, indexation et transformation dans le système, illustré par la courbe rouge, est de 320 ms. En conséquence, après une moyenne de 450 ms toutes les communications sont terminées et ceci permet de mettre en évidence les bonnes performances du système.

4 Sécurité hybride pour les réseaux corporels

L'amélioration du niveau de vie, la croissance des coûts des soins et le vieillissement des populations d'une part et le développement des puces électroniques et des communications sans fil d'autre part ont permis le développement de nouveaux types de réseaux sans fil, appelés réseaux corporels (WBAN) pour surveiller des patients à distance. Les WBAN présentent certaines similitudes avec les réseaux de capteurs sans fil telles que la faible capacité de calcul, de mémoire et d'énergie, mais aussi une quantité importante de données générées et nécessitant une bonne gestion et un bon traitement. Le middleware iSensors est donc particulièrement bien approprié pour gérer ces données.

Un réseau corporel (WBAN) est constitué essentiellement de capteurs ou d'actionneurs intelligents implantés ou attachés au corps des patients et qui utilisent une communication sans fil avec une unité de traitement locale et personnelle (LPU). Ce réseau assure alors la transmission des données personnelles des patients, qui sont à la fois privées et sensibles. Ainsi, en plus de la gestion des données et afin de préserver l'intimité et la santé des patients, le *middleware iSensors* doit assurer leurs sécurisation.

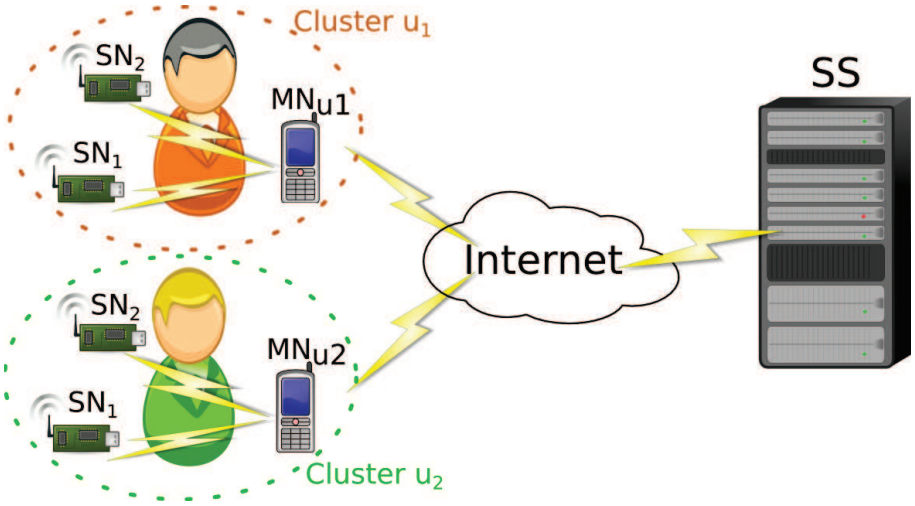


FIGURE 5 – Architecture d'un WBAN.

En tenant compte de cette problématique de sécurité et de l'architecture du réseau WBAN, représentée en Fig. 5, un protocole hybride de sécurité a été proposé. Les SN sur la figure représentent les capteurs d'un utilisateur alors que les MN jouent le role de LPU. SS est le site de stockage. Sur la figure, deux types de communications entre les différents composants de l'architecture sont représentés. Le premier type, nommé la communication "intra-corporelle" (*Intra-Body Communication* ou IbC) représente la communication des capteurs dans la grappe où les ressources sont restreintes. Le deuxième type, nommé "communication extra-corporelle" (*Extra-body Communication* ou EbC), représente la communication entre le MN et le SS qui, quant à eux, disposent de plus de ressources. En plus de leurs différences au niveau des ressources, IbC et EbC rencontrent des menaces de sécurité différents. En effet, au niveau IbC, les SN et leur MN associé sont attachés à un seul utilisateur, ce qui fait de l'intrusion d'un attaquant une tâche difficile. En outre, le petit rayon dans lequel le Ibc opère (environ 2 m) facilite la sécurité car il faut être proche du réseau de capteurs pour attenter à sa sécurité. La sécurité au niveau EbC est plus délicate, surtout si l'on considère que les communications entre MN et SS s'effectue au travers de l'Internet, ceci ayant pour effet de multiplier les risques ainsi que les types d'attaques.

Considérant que IbC et EbC ont des caractéristiques et des exigences sécuritaires différentes, nous proposons un protocole hybride d'authentification et d'établissement de clés basé sur deux protocoles, le premier gèrant l'authentification mutuelle entre SS et MN, alors que le second traite l'authentification et l'établissemnet d'une clé de groupe entre SN, MN et SS. La propriété hybride est utilisée pour mettre en évidence le fait que les clés asymétriques sont générées et utilisées pour signer des messages au niveau EbC, tandis que des clés symétriques sont générées et utilisées au niveau IbC.

4.1 Schéma d'authentification et d'établissement de clés entre MN et SS

Ce premier protocole, traitant de l'authentification dans EbC repose sur la cryptographie basée sur les identités (IBC), initialement proposé par Adi Shamir en 1984 [9]. En effet, les entités dans EbC ont besoin d'un haut niveau de sécurité car ils communiquent par le biais de l'Internet et disposent de ressources suffisantes pour exécuter les fonctions fournies par IBC. L'objectif de ce protocole est d'effectuer l'authentification mutuelle entre SS et MN, en fournissant une paire de clés asymétriques pour MN en utilisant le schema de Hess [10], et en établissant une clé de paire partagée entre MN et SS, générée en se basant sur les hypothèses du problème de Diffie-Hellman (C-DH).

4.2 Schéma d'authentification et d'établissement de clé entre SN-MN-SS

Ce deuxième protocole concerne l'authentification au niveau IbC. En raison des ressources limitées des entités, seules des clés symétriques peuvent être générées et utilisées. En effet, après avoir établi un lien sécurisé entre MN et SS, grâce au protocole défini ci-dessus, les nœuds SN doivent s'authentifier à la fois à leur MN associé, ainsi qu'à SS, avant d'être en mesure de communiquer avec eux. Le protocole présenté pour IbC vise à fournir un mecanisme de sécurité permettant à SN d'établir une clé tri-partites et deux clés de paires, chacune étant partagée avec MN et SS respectivement. À la fin de cette phase, SN établit des clés de paires avec MN et SS, ainsi qu'une clé tri-partites basée sur le protocole *Joux key agreement* [11].

5 Un réseau social pour les RCSF

L'apparition des réseaux sociaux a révolutionné le monde de l'Internet. En effet, les applications de partage de données, d'avis ou d'images ont rencontré un grand succès auprès des utilisateurs et des organismes. Un tel partage, déployé dans les réseaux RCSF, peut aussi être très bénéfique. En effet, il ouvre la porte aux chercheurs, par exemple, à une collaboration étendue où chaque laboratoire met à dispositions des autres les informations collectées par ses réseaux. Ainsi, ce partage peux servir dans le domaine de l'informatique mais aussi de la météologie, du médical, le monde universitaire, etc.

Étant conscient de l'enjeu de ce partage, le *middleware iSensors* présenté initialement ne peut être utilisé que par un seul utilisateur ou organisation. Dans cette partie de thèse, il est étendu avec de nouvelles fonctionnalités multi-utilisateurs et de partage. Afin de réaliser le portail social pour les réseux de capteurs sans fils, trois extensions ont été proposées. La première consiste à rendre le *middleware* accessible de la façon qu'un service sur le *cloud* pour qu'il devienne multi-utilisateurs. La seconde consiste à étendre les composites du *middleware* avec des gadgets qui peuvent être intégrés dans le portail de l'organisation, de l'utilisateur ou dans le portail d'un tiers. Ces gadgets servent à partager les données des capteurs et à offrir par la suite un portail social pour les données des réseaux de capteurs. La troisième extension consiste à permettre aux utilisateurs de contrôler l'accès à leurs données.

5.1 L'infrastructure *Cloud* pour les réseaux de capteurs

Afin de réaliser la fonction multi-utilisateurs, l'architecture illustrée en Fig. 6 a été proposée. Elle a été conçue comme un logiciel fournissant un service (*Softaware as a Service : SaaS*) dans une infrastructure de type *Cloud Computing*, afin d'offrir un service de gestion des données des capteurs pour de nombreux utilisateurs. Ainsi, chaque utilisateur

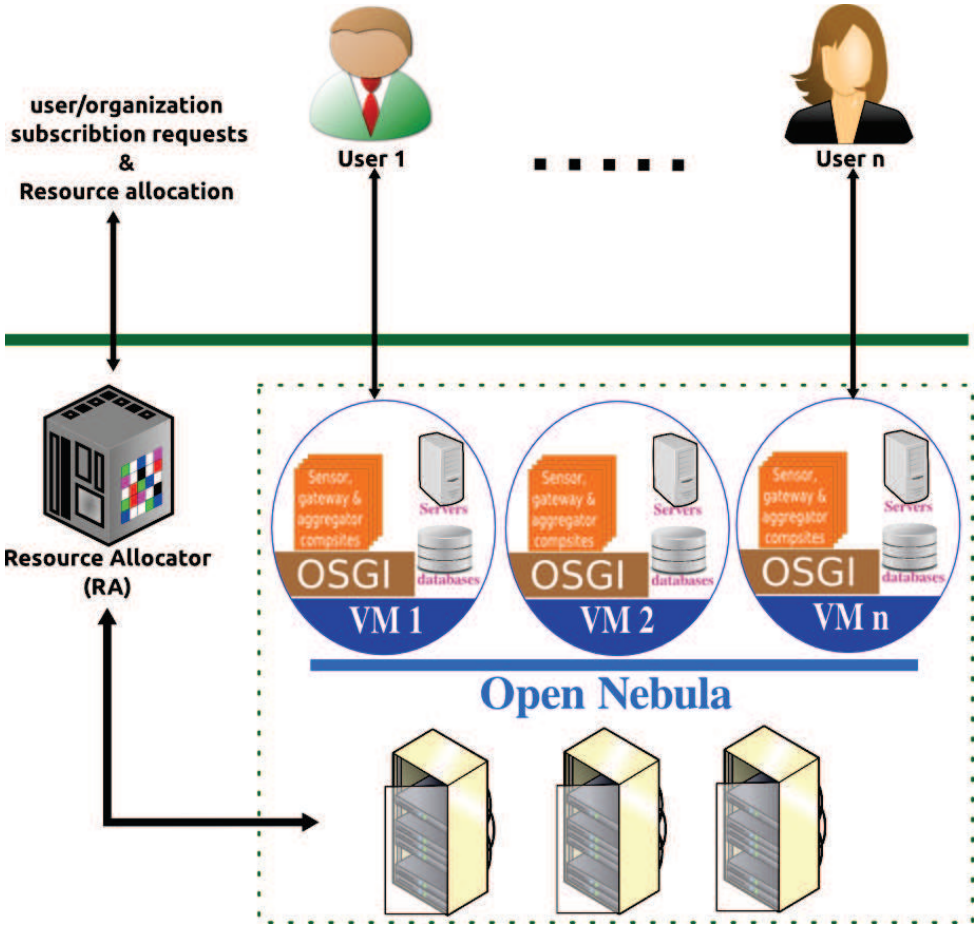


FIGURE 6 – Architecture globale de l'infrastructure cloud.

possède son propre système qui lui permet de gérer ses données issues des capteurs. Un déploiement d'instances séparées du *middleware* pour chaque utilisateur dans le *cloud* a été retenu pour différentes raisons :

- la possibilité de connecter différents périphériques et réseaux ;
- le stockage des données et des index dans le *cloud* ;
- la facilité d'ajout et de suppression de nouveaux utilisateurs/organisations ;
- l'utilisation des services sans se soucier de l'infrastructure ;
- la haute performance et la haute disponibilité.

La figure montre que chaque utilisateur possède sa propre machine virtuelle (VM) qui héberge son *middleware* ainsi que d'autres serveurs nécessaires tels que le serveur de stockage et celui des événements. La création d'une nouvelle machine virtuelle est effectuée par le serveur d'allocation de ressources (RA) à réception d'une requête de souscription d'un nouvel utilisateur. La VM est une instance d'un modèle (*template*) pré-défini. Elle est ensuite personalisée en fonction de la demande de l'utilisateur. Après la création de la VM, l'utilisateur peut s'y connecter afin de l'administrer et de la personnaliser. De plus, le RA peut supprimer et mettre à jour les VM existantes.

5.2 Le réseau de capteurs social

Grâce à l'infrastructure *cloud* pour les RCSF, les données des capteurs de différents utilisateurs sont isolées dans des bases de données séparées et hébergées sur des machines virtuelles propres à chaque utilisateur/organisation. Afin de permettre la collaboration et le partage de ces données, le *middleware* iSensors et l'infrastructure *cloud* ont été étendues avec les fonctionalités des réseaux sociaux afin de définir un portail web social pour les RCSF. Une telle extension présente de nombreux avantages :

- le partage des données issues des capteurs ;
- l'utilisation de gadgets (par défaut/personnalisés) inclus dans les composites du *middleware* ;
- la surveillance en temps-réel ;
- des mécanismes de sécurité différents.

Dans cette extension, illustrée en Fig. 7, le composite a été étendu avec un gadget afin d'afficher les données de ce composite et permettre la navigation dans l'historique. Ces gadgets permettent de fournir des interfaces utilisateurs pour visualiser les données des capteurs en temps-réel et leur historique à l'aide de tableaux ou de graphiques. Chaque utilisateur doit disposer d'un portail personnel dans lequel il met ses gadgets, mais aussi des gadgets importés d'autres utilisateurs, qui sont ses amis. Pour gérer les liens entre les utilisateurs l'API sociale est utilisée. De plus, une autre API a été ajouté pour le capteurs sociaux afin de stocker des informations sur les gadgets de l'utilisateur et les utilisateurs de ses propres gadgets.

5.3 La sécurisation des accès dans le portail social

Comme présenté ci-dessus, le portail social des capteurs permet de partager des données entre les utilisateurs. En conséquence, les amis d'un utilisateur U peuvent voir toutes ses données peuvant contenir des informations privées et personnelles. Dans ces circonstances, garantir un accès contrôlé aux données de l'utilisateur est devenue une nécessité impérieuse afin de protéger sa vie privée. Pour cette raison, nous proposons un schéma de sécurité basé sur la cryptographie avec des attributs (Attribute-Based Encryption) [12] pour protéger les données partagées des utilisateurs. Ainsi, l'utilisateur U peut crypter les données tout en associant une combinaison booléenne d'attributs à chacuns des types de données. Seuls les utilisateurs ayant le bon ensemble d'attributs résolvant la combinaison peuvent décrypter ce type de données. En fait, l'utilisateur U fournit à chacun de ses amis une clé asymétrique contenant un ensemble d'attributs que U choisit pour définir les droits accordés à cet ami.

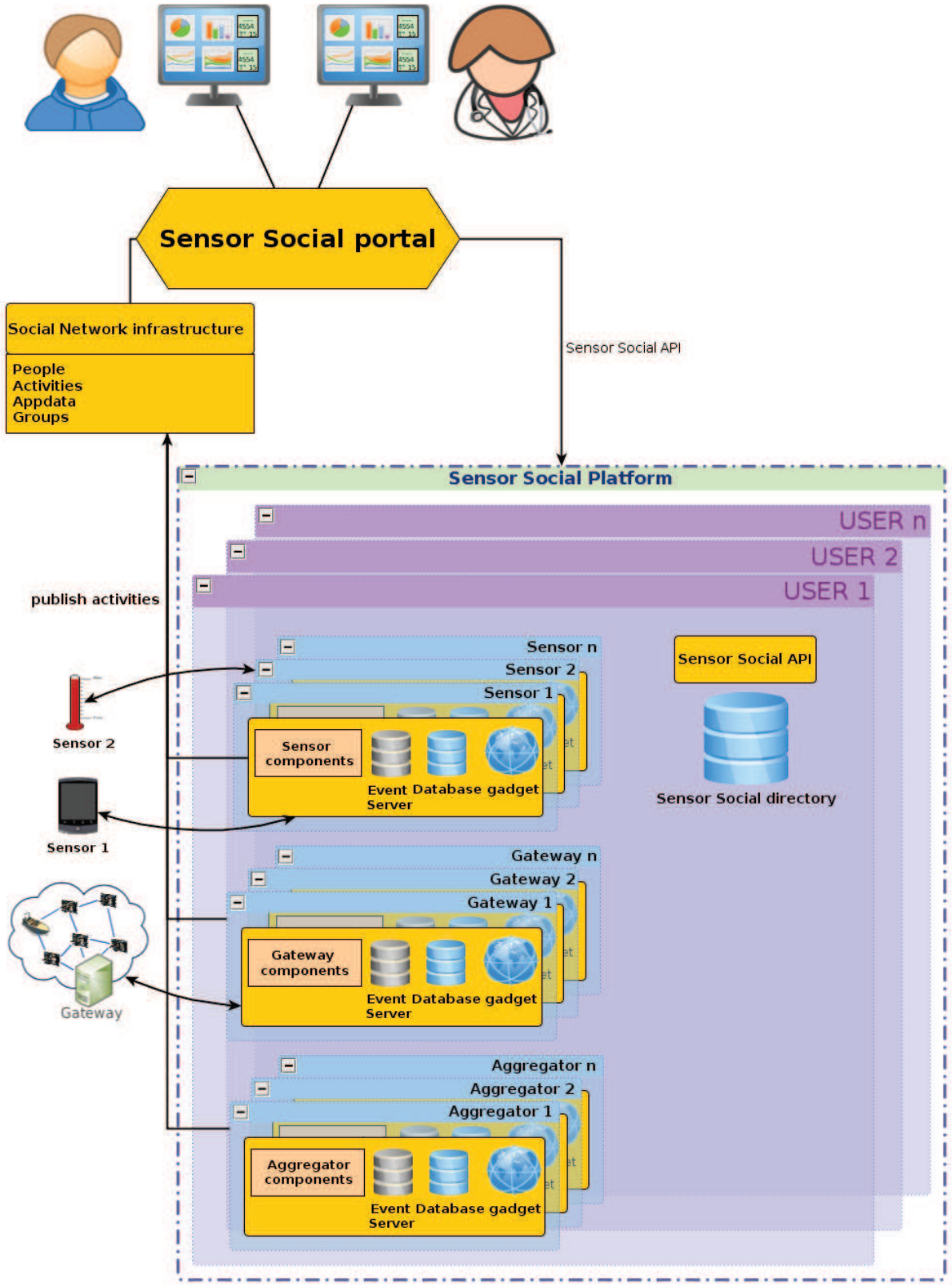


FIGURE 7 – Architecture globale des réseaux de capteurs sociaux.

6 Conclusion

Dans cette thèse nous nous sommes intéressés à la collecte sécurisée des données dans les RCSFs, la gestion et l'analyse de ces données et leurs partage sécurisé dans un portail social dédié aux RCSF.

Le *middleware iSensors* a été proposé afin de gérer la réception, le stockage, l'indexation et l'aggrégation des données reçues des capteurs hétérogènes, ainsi que la génération d'alertes en cas de besoin. Pour pallier les limites des capteurs, nous nous sommes intéressés à la production d'un *middleware* permettant de stocker et traiter les données dans des *serveurs extérieurs* aux RCSF. L'utilisation de serveurs de haute performance a donc permis d'obtenir de meilleurs temps de réponse aux requêtes des utilisateurs. En outre, ceci ajoute de la souplesse au *middleware* grâce au fait que ces serveurs n'exigent pas un langage ou un programme spécifique à charger dans les capteurs. Ainsi, il est facile de relier de nouveaux noeuds dans les RCSF sans nécessité d'y installer de nouvelles applications.

Le *middleware iSensors* a été mis en œuvre dans le cadre du projet européen Mobesens qui vise à surveiller la qualité de l'eau. La flexibilité et la dynamicité du *middleware* ont permis son adoption et interfaçage avec les RCSF déployés dans Mobesens ainsi qu'avec une application de visualisation des données en temps-réel.

Afin de sécuriser le RCSF et empêcher des utilisateurs indésirables d'introduire des nœuds malveillants dans le système, et ainsi falsifier ou récupérer des données, un schéma d'authentification hybride et d'établissement de clés se déroulant en deux phases a été mis en place. Il a été conçu pour sécuriser les transferts de données du RCSF vers le *middleware* en utilisant deux niveaux de sécurité en fonction des capacités des nœuds participants. Une sécurité avec des clés asymétriques a été adoptée entre la passerelle et le *middleware*, tandis que des clés symétriques ont été utilisées pour sécuriser les deux liens entre le capteur, et la passerelle et le *middleware*.

Étant donné l'importance du partage des données collectées et l'étendu de l'utilisation des réseaux sociaux de nos jours, le travail de la thèse conclut par la proposition d'un portail social sécurisé permettant le partage de données issues de capteurs. Cette solution n'est pas intégrée dans un site social existant. Cependant, elle est réalisée grâce à l'extension des composites à base de gadgets génériques ou personnalisés qui peuvent être intégrés dans les portails de différents utilisateurs. Ainsi, chaque utilisateur gère son propre portail en personnalisant les gadgets qui y figurent et la liste de ses amis (qu'ils acceptent ou refusent suite à une demande d'ajout). De plus, il peut restreindre l'accès aux données de ses capteurs en utilisant une solution de sécurité pour la définition des droits d'accès à l'aide de la cryptographie basée sur les attributs.

Bibliographie

[1] eXist-db, the Open Source Native XML Database. http://exist.sourceforge.net.

[2] XMPP protocol. http://xmpp.org/.

[3] Java Message Service (JMS). http://java.sun.com/products/jms.

[4] Mobility for Long Term Water Quality Monitoring. http://www.mobesens.eu/.

[5] J. Rousselot, Ph. Dallemagne, and Decotignie J-D. Deployments of wireless sensor networks performed by csem. In *COGNITIVE SYSTEMS WITH INTERACTIVE SENSORS*, 2009.

[6] A. El-Hoiydi and J.D. Decotignie. Wisemac : An ultra low power mac protocol for multi-hop wireless sensor networks. In *ALGOSENSORS*, pages 18–31, 2004.

[7] A. El-Hoiydi, C. Arm, R. Caseiro, S. Cserveny, J.-D. Decotignie, C. Enz, F. Giroud, S. Gyger, E. Leroux, T. Melly, V. Peiris, F. Pengg, P.-D. Pfister, N. Raemy, A. Ribordy, D. Ruffieux, and P. Volet. The ultra low-powerwisenet system. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1 –6, march 2006.

[8] Tahoe-LAFS. http://tahoe-lafs.org/trac/tahoe-lafs.

[9] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[10] F. Hess. Efficient identity based signature schemes based on pairings. In Kaisa Nyberg and Howard Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer Berlin / Heidelberg, 2003.

[11] A. Joux. A one round protocol for tripartite diffie-hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[12] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

Chapter 1

Introduction

Wired sensor networks have been very used to monitor machines, factories and homes. Their limitations are due to the use of wires which can make their installation and interconnexion a cumbersome task immediately with the growth of the number of nodes or topology change. Hence, Wireless Sensor Networks (WSNs), which are considered as the result of the advances realized in micro-electromechanical systems technology, digital electronics and wireless communications over the last fifteen years, come to resolve the aforementioned issues. In contrast, they raise new issues and challenges.

1.1 Wireless Sensor Networks

WSN has introduced new components named sensor nodes. These tiny and autonomous communicating nodes, which have the ability to monitor one or more physical phenomena, are self configured and build a wireless network with their neighbors at boot strap. Usually, a network is composed of a multitude of homogeneous or heterogeneous sensor nodes that form a mesh network, and there is one or more sinks in the network border to collect data and/or interconnect the network to other networks such as the Internet. The ease of use, the deployment and the organization of these networks pave the way for novel applications that were impossible with legacy technology. This explains the wide number of WSN applications used in various domains such as in battle fields to gather critical data, in health care services to better manage emergency situations or for monitoring both air or water quality.

However, like many other wireless components, these tiny nodes are characterized by their resource limitations in terms of energy, bandwidth, memory and computation capacity. This affects the whole network and poses new challenges for researchers and industrials to ensure the efficiency, reliability, security, generated-data management and self-configuration.

In order to handle the aforementioned challenges, specific applications should be de-

veloped for sensor nodes, respecting their caracteristics, to be run on top of a specific Operating System (OS) or in a middleware. Usually, WSNs are application specific. However, in recent years and thanks to new programing trends, some techniques like virtualization and cloud computing have been more and more used to provide flexible and reconfigurable sensor networks. Nevertheless, there are many application-specific networks that are already deployed and in use.

Due to their specific nature, wireless networks are more vulnerable to external attacks and present more security breaches as compared to wired networks. The WSN security is also a critical concern which should not be neglected. In fact, nodes are usually deployed without protection neither surveillance. Consequently, they encounter many physical threats such as the tampering of the node, the injection of a fake node or their replication.

1.2 Data management in WSNs

Usually, the WSN is composed of a multitude of homogeneous or heterogeneous sensor nodes that collects measurements at different frequencies. A huge amount of data is consequently generated in the network and needs to be translated and analysed in order to be interpreted and deduce alerts, if necessary.

Further to the data collection, storing them in the WSN is a very delicate task especially as the sensor node's storage capacity is very limited. To overcome this problem, the use of an external storage is also conceivable. Also, a tradeoff between the two storing approaches can be realised. Nevertheless, when data must be kept for a long period, the use of an external storage becomes an obligation, which enhances the system especially with the technological advancement of databases and cloud computing.

Users or applications can collect these data from the network by sending queries or program routines to retrieve periodic updates or raised alerts. As data can come from heterogeneous sensor networks, the collecting system should take into account this heterogeneity and should understand all data inputs in order to be able to well process them. In fact, data processing means execution of some operations such as analysis, aggregation, generation of statistics (average, maximum, etc.), conversion or translation of the data. Processing can be performed in the node, the network, as well as in an external system.

This network enables to open innovation opportunities and to create various applications which can benefit from the availability of these data. The web-social sharing of data is one among them. Indeed, geographically distant researchers sometimes wish to work together by comparing and analyzing their sensor data and their results. Scientists in different fields may also collaborate to interprete a physical phenomenon, or doctors in diverse countries can diagnose a patient having implanted sensors that collect some vital signs. This data sharing in a web-social application is highly beneficial for the rapid advancement of research and has become feasible thanks to data availability and the ease

of access. In short, the collection, management, securisation, exploitation, and sharing of sensor data represent the real challenges for WSNs.

1.3 Contributions

The widespread use of WSNs in a multitude of application domains and the heterogeneity in techniques has led to an increasing complexity for control and management of data they collect or report because the application sector use and rely on very specific solutions. This is exacerbated by the fact that each person or organization possesses a multitude of heterogeneous devices and sensors. This thesis addresses this complexity by proposing new approaches to handle heterogeneity when collecting, managing and sharing sensor data.

The first contribution adresses the collection and management of data produced by heterogeneous WSNs. Previous works are application-specific, require specific data formats or require updates of programs in already deployed sensor nodes in the field. A component-based middleware named iSensors that provides a standardized and lightweight RESTful interface to collect data from SNs or gateways has been proposed to remove these hurdles. In order to provide the different services of the middleware, default components are defined to setup a private composite per node in communication with the middleware. Each composite is a collection of components used to receive, store, index, translate and process sensor data. Data processing is intended to add new meta-data, like data validation, or generate alerts which are routed using an eventing manager component. Users can personalize or control the behavior of their components using xQuery scripts. To make the middleware even more flexible OSGi is used to host components and composites, no-SQL database is used to store any data format, XML files are indexed to run efficient queries and messaging services are used to route alerts between composites or to external applications.

The second contribution consists of a real-time monitoring and management support system for water quality monitoring WSNs in the context of the Mobesens project. A visualization system is in charge of providing web interfaces to scientists and users to monitor and configure the WSN and visualize measurements in real-time. Upon receiving sensor data from a gateway, the middleware generates alerts which are intercepted and displayed in real-time. The visualization system also uses the RESTful API of the composites to get and search history data to be processed and displayed. Thus, specific use case confirmed the flexibility of the proposed middleware. To handle long term storage of collected data a new component was added to the middleware to store data in cloud storage system. The back-end middleware and the visualization system architecture is reported.

As stated above, security is a major concern in WSNs especially when private and

sensitive information are transported as in healthcare monitoring applications. The issue of securing communications between sensor networks and the middleware is tackled in the **third contribution** via a hybrid security scheme to address heterogeneity of compute, memory and energy characteristics. The proposed authentication and key agreement scheme uses symmetric cryptography in resource limited sensor/actuator nodes and identity-based cryptography between the gateway and the middleware that have more abundant resources. We present two protocols to authenticate and establish pairwise and group keys between all tiers and to provide public and private keys for the gateways.

Sharing sensor data is often needed in applications to enable knowledge sharing and to enhance collaboration. The iSensors middleware was originally designed to handle sensor data of only a single user or an organization. To overcome this limitation, the **fourth contribution** extended the system with a secure social sensor portal to enable data sharing between users or organisations. Previous research focussed on how to integrate sensor data in existing social networks. In our case, we propose a dedicated social portal for sensor networks using web gadgets in each composite to facilitate the access and control of sensor nodes as well as sharing sensor data between users. The contribution is presented in three steps. First, a cloud infrastructure is proposed for sensor networks where users can have their own servers equiped with a middleware. Second, the middleware is extended with social network capabilities to provide a social sensor portal which enables data sharing between users by facilitating mashups sharing. Users can integrate their mashups or those of their friends in their portal. Since data sharing in the social sensor portal raises new privacy challenges, we provide a new security scheme using Attribute-Based encryption to enable users control what other users are authorized to see and access.

1.4 Outline

The rest of the thesis is organized in six chapters, as follows. Chap. 2 provides background information on characteristics of the WSN technology and their impact on data management. In the first part, we analyze hardware platforms of Sensor Nodes (SNs) and their impact on the network characteristics. Next, we enumerate some applications of WSNs. Then, we discuss and compare some OS properties. In the second part, we discuss data management challenges in WSNs and their different levels while presenting the most important solutions in the literature. Finally, convergence between WSNs and both cloud computing and social networking is discussed in parts three and four respectively.

In Chap. 3, we introduce the main features and components of the proposed middleware. Here, we provide arguments for the use of the component model, focusing on its flexibility and configurability. We present the middleware architecture and implementation. Then, we suggest some extensions.

Chap. 4 presents a use case of the proposed middleware in the context of the Mobesens European project. First, the project and its requirements are presented. Then, the architecture and the used techniques to manage the collected data from SNs are described. The chapter concludes on the description of some real-world experiments conducted during the project and by evaluating the performance of the implemented system.

In Chap. 5, a mutal authentication scheme between SNs, the gateway and the middleware is presented in order to secure their communications. First, we highlight the need for security and the differences between involved elements in the scheme. Then, we provide the related security background. Next, different steps of the scheme are explained and discussed. Finally, we analyse the scheme resiliency against certain threats and we discuss the performance.

Chap. 6 presents a secure social sensor portal for sharing sensor data between users. To realise the portal three features are needed. First, a cloud infrastructure for sensor networks to provide an instance of the middleware for each user. Second, a social sensor network extension to the middleware to let it provide social services. Third, a security scheme to protect the user's sensor data.

The dissertation concludes in Chap. 7 with a summary of the contributions and several directions for possible future work.

1.5 Publications related to this thesis

This section lists the publications made during the preparation of the thesis.

- Wassim Drira, Eric Renault, and Djamal Zeghlache. "Design and performance evaluation of a composite-based back-end system for WSNs". *In the 8th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC), 2012 ,* august 2012. ISBN 978-1-4577-1379-8. [Related to Chap. 4]

- Wassim Drira, Eric Renault, and Djamal Zeghlache. "A hybrid authentication and key establishment scheme for WBAN". *In the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom),* pages 78 –83, june 2012. [Related to Chap. 5]

- Wassim Drira, Eric Renault, and Djamal Zeghlache. "iSensors: A middleware for dynamic sensor networks". *In the 2nd International Conference on Communications and Information Technology (ICCIT),* pages 134 –138, june 2012. [Related to Chap. 3]

- Wassim Drira, Eric Renault, and Djamal Zeghlache. "Towards a dynamic middleware for wireless sensor networks". *In the 2nd International Conference on Net-*

working and Future Internet (ICNFI), 2012, pages 42 –45, april 2012. ISBN-13: 978-2-915618-24-2. [Related to Chap. 3]

- Wassim Drira, Eric Renault, and Djamal Zeghlache. "Design and performance evaluation of a system for storing and visualizing data from a sensor network". *In the 4th International Conference on Electronics Computer Technology (ICECT), 2012,* volume 3, pages 50 –54, april 2012. ISBN 978-1-4673-1849-5. [Related to Chap. 4]

- Eric Renault, Wassim Drira, Houssem Medhioub, and Djamal Zeghlache. "Management and semantic description of objects for the future internet". *In the 2nd International Conference on Ubiquitous and Future Networks (ICUFN), 2010,* pages 291 –296, june 2010. [Related to Chap. 3 and Chap. 4]

Chapter 2

State of the art

Contents

Over the last decade, significant advances have been made in Wireless Sensor Networks (WSNs), networking, computing, managing and data sharing. A convergence between these fields is needed to overcome the limitations of WSNs and improve provided services.

This chapter provides in the first part an overview of the WSN technological landscape, properties, applications and data management challenges. The second part describes different levels of data management in these networks and surveys some existing solutions.

Finally, convergence between WSNs and both cloud computing and social networking is discussed in part three and four respectively.

2.1 Wireless Sensor Networks : background and challenges

The advances in micro-electromechanical system technology, digital electronics and wireless communications have enabled the development of low-cost sensor nodes that communicate in short distance, and can process and store data. Nodes for industrial use may be very small, as presented in Fig. 2.1, due to the application of specific miniaturization process when big quantities are produced. However research sensor nodes are bigger, for example the tmote sky [1] size is $6.6 \times 3.27 \times 0.7$ cm. These nodes have allowed the development of a new kind of networks also known as WSNs which currently have many applications.

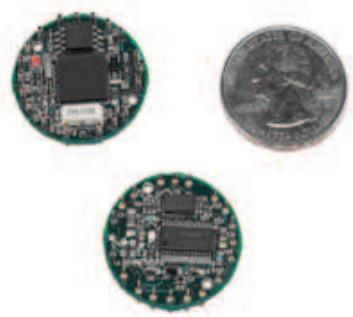


Figure 2.1: Dust sensor nodes.

2.1.1 Hardware Platforms

Research sensor nodes are numerous and have different specifications. However, they share some properties and constraints. They usually have a processing unit in the form of a microcontroller which includes a processing unit, a program memory and a Random Access Memory (RAM). They also have a limited source of energy as they operate with the help of two AA batteries. Sensor nodes communicate wirelessly using a small low-cost and low-power communication chip.

A set of the most used platforms is described in Table 2.1. With the exception of IMote2, all nodes in Table 2.1 have very limited computing and memory resources. The tmote sky [1], presented in Fig 2.2, is one of the most widely used platform. Its microcontroller has a clock speed of 8 MHz, a maximum program size of 48 kB and a RAM size of 10 kB. In addition, its communication operates at a bit rate of only 250 kbps using the

IEEE 802.15.4 standard [2] which specifies the physical layer and media access control for low-rate wireless networks.

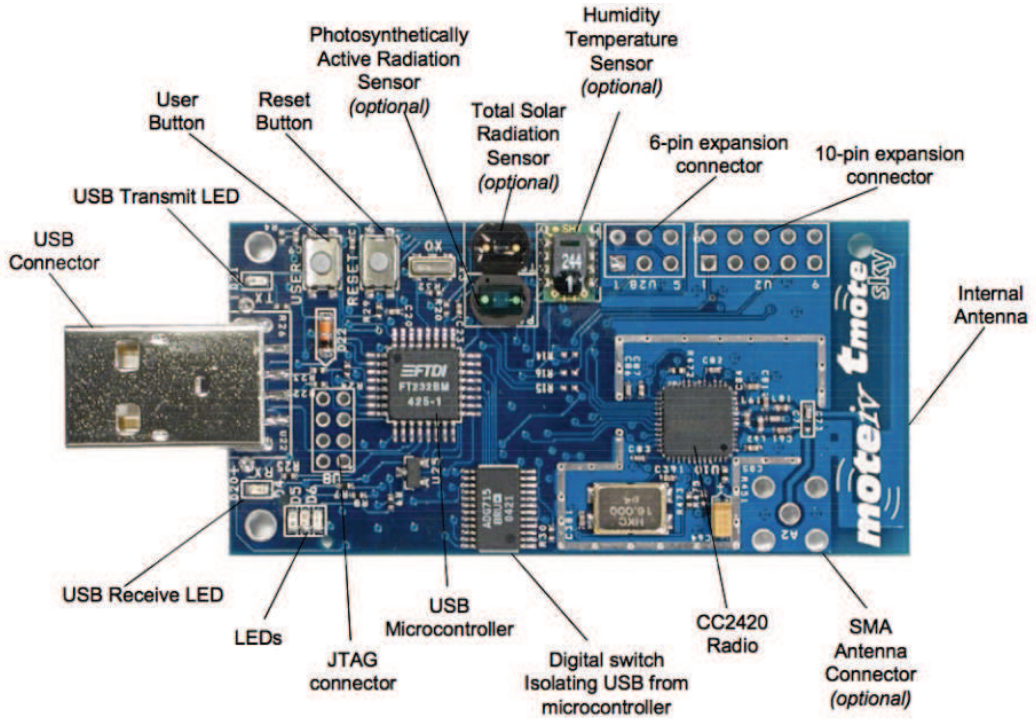


Figure 2.2: Anatomy of the tmote sky platform.

Moreover, sensor nodes have sensing elements for gathering contextual information from the physical reality around it. The nature of the collected information and the type of these elements strongly depend on the application in which the node is used. The tmote sky may integrate humidity, temperature, and light sensors.

Both size and energy constraints put limits on the communication system which operates mostly using small data packets sent over relatively short distances and with low average data rates. The power consumption of the node is dominated by the radio electronics and protocol solutions that limit the amount of idle listening are needed to achieve long system lifetimes.

Despite these constraints, sensor nodes enable the construction of large distributed monitoring and actuation networks that are deeply embedded in the physical environment and offer unprecedented levels of temporal and spatial sampling density. Thanks to their low-cost, tiny size, wireless communication, easy deployment and maintenance cost, they open new challenges and pave the way for novel applications that were either impractical or impossible with legacy technologies.

Platform	MicaZ [3]	telosb [4]	Tmote Sky [1]	BTnode3 [5]	SunSPOTv1 [6]	IMote2 [7]
Producer	Crossbow	Moteiv Corporation		ETH Zurich	Sun Microsystems	Intel
Release	2002	2004	2004	2006	2005	2005
Microcontroller						
Type	Atmel ATmega128L	TI MSP430	MSP430	Atmel ATmega128L	Atmel AT91RM92	Intel PXA271
Frequency	8 MHz	4 MHz	8 MHz	8 MHz	25 MHz	520 MHz
Program Memory	128 KB	48 KB	48 KB	128 KB	128 KB	-
RAM	4 KB	10 KB	10 KB	4 KB	16KB	32 MB
Storage	512 KB	1 MB	1 MB	3x60 KB	4 MB	32 MB
Communication						
Type	Chipcon CC2420	Chipcon CC2420	Chipcon CC2420	Zeevo ZV4002/ Chipcon CC1000	Chipcon CC2420	Chipcon CC2420
Bit rate	250 kbps	250 kbps	250 kbps	721/76.8 kbps	250 kbps	250 kbps
Expansion	51-pin	USB & 10-pin	USB & 16-pin	UART	USB	USB & RS232

Table 2.1: Main platforms and their main components.

2.1.2 Network characteristics and challenges

The use of tiny sensor nodes having limited resources makes WSNs different from traditional wireless networks. These different characteristics bring new challenges to network designers [8, 9].

- ***Deployment :*** Sensor nodes can either be deployed randomly or precisely, and their number can vary from some nodes to many. The deployment method is closely related to the network application and cost nodes. In [10], authors survey the most prominent examples of sensor network deployments. The density and the number of these nodes in the network affect scalability, reliability, accuracy, data management protocols and the degree of coverage of the area of interest.

- ***Energy consumption :*** Sensor nodes usually operate with the help of batteries which are difficult to change or reload when nodes are deployed in an unattended area. Hence, the life time of sensor nodes and therefore of the whole network strongly depends on the battery lifetime.

- ***Limited hardware resources :*** As presented in Sec. 2.1.1, sensor nodes suffer from resource limitations such as memory, computation, processing, communication, etc.

- ***Application specific :*** A WSN is deployed to perform a specific task, so only needed binaries and librairies are loaded into the node. So, after deployment, it is not obvious to use them for another application. Some applications are described in Sec. 2.1.3.

- ***Unreliable sensor nodes :*** Sensor nodes are deployed in harsh or hostile environment which makes them prone to failures and physical damages.

- ***Self-configurable :*** Sensor nodes must be autonomous and be able to configure themselves into a communication network. Moreover, the network may be either mobile or static, and nodes can appear and disappear frequently due to node failures, damage, addition, energy depletion, or channel fading. To allow continuous and reliable operation of the network, nodes must self-configure to meet these constraints.

- ***Data Aggregation/Data Fusion :*** The sensor data sensed by multiple sensor nodes typically have a certain level of correlation or redundancy due to the continuity of the physical environment and the density of the network. Thus, data aggregation and fusion techniques could be used to reduce network traffic.

- ***Many-to-one and one-to-many traffic pattern :*** In general, the sensor data captured by a sensor node is transmitted to a gateway, a sink or a base station through one or multiple hops, which exhibits a many-to-one communication pattern. A sink can broadcast a command, new parameters, or a query to all nodes, which exhibits a one-to-many communication pattern.

- ***Quality of Service (QoS) support :*** This can be integrated into all network layers, and can be a cross-layer service [11]. QoS requirements, such as delivery latency, packet loss, freshness, fidelity, coverage and reliability, etc. depend from an application to another. Moreover, QoS is related to the accuracy and frequency of sensor measurements.

- ***Security :*** WSNs suffer from many threats which are described in [12] and summarized as : passive information gathering, node subversion, fake node injection, node malfunction, node outage, node replication, message corruption, traffic analysis, routing loops, selective forwarding, sinkhole attacks, sybil attacks, wormhole and deny of service attacks. In addition, WSNs have limited hardware and energy resources. Thus, traditional security schemes cannot be directly applied. Security

requirements are closely related to applications, so designers of security protocols should make a compromise between limitations and requirements.

2.1.3 Network applications

Wireless communication, tiny programmable nodes and the availability and low-cost of sensors of different physical phenomena make WSNs easy to deploy and use. Many projects have completed or are in progress and this section presents the most relevant applications while focusing on some of them related to our work [13, 12].

2.1.3.1 Military

WSNs can be used to supervise a country frontier, to track enemy soldiers and to detect and localize enemy snipers and soldiers. A self-healing land mines can use a network of antitank landmines to monitor threats to nodes in order to ensure that a particular area remains covered even if the enemy tampers with a mine to create a potential breach lane [14]. A remote chemical biological, and explosive vapor detection system can be used to measure trace concentrations of explosives, toxic chemicals, and biological agent signatures [15].

2.1.3.2 Emergency situations

A monitoring network for automatic fire detection can be used for early detection, promptly extinguishing fire and rescuing people quickly. These networks can exist in residential areas or in forests, and some related works have been surveyed in [16]. A WSN can be used to monitor rivers and generate early flood warning [17]. Moreover, sensor networks can be deployed when disasters occurred in the specific area to monitor the situation as they are autonomous, self-configurable, cheap and communicate wirelessly.

2.1.3.3 Buildings

WSNs have various aplications to provide smart homes. They can be used to save energy [18], to control heating and cooling systems or earthquake mitigation [19, 20].

2.1.3.4 The smart power grid

The advances in sensor networks have paved the way to modernize the power grid and provide more services in addition to electricity distribution [21]: *a*) allow two-way flow of both information and electricity, *b*) be self-healing, *c*) improve energy storage, *d*) be environment friendly by minimising emissions of greenhouse gases, *e*) allow real-time pricing, and *f*) manage residential energy.

2.1.3.5 Smart bridges and smart tunnels

Sensor networks can be deployed in bridges and tunnels to monitor their operational performance and deterioration [22]. Moreover, it can be used to monitor traffic in tunnels and detect accidents in order to reduce damages [23].

2.1.3.6 Vital signs

As presented in [24, 25], sensor nodes are used to monitor many vital signs like temperature, heart rate, breathing rate, blood pressure, electrocardiogram, etc. Reports are sent to the doctor and when alerts are generated, they are transmitted to an emergency rescue team.

2.1.3.7 Supply chain

WSNs can be used a monitor the cold chain and supervise the goods temperature along transportation from the manufacturer to the consumer [26].

2.1.3.8 Environment

Environment monitoring has been the concern of many works. The GlacsWeb project [27] monitors the behavior of ice caps and glaciers for understanding the earth's climate. Collected data from probes (inserted in the glacier) are relayed through Base Stations to a Sensor Network Server where it becomes accessible using web services. Jung et al. [28] proposed an air pollution monitoring system involving a context model for understanding the air pollution status on the remote place. The Integrated Sensor Web Grid Cyberimplementation [29] has two goals. The first one is to demonstrate the integration features of heterogeneous environmental wireless sensor networks. The second one is to present a sensor web grid Cyberimplementation as a server hiding the system heterogeneity.

2.1.3.9 Water quality

Other projects deal with the monitoring of water quality. Wang et al. [30] proposed a Remote Water Quality Monitoring System based on WSN. In their system, sensors send measurements data via a coordinator to a control center where they are stored in an SQL database and shared with other users. Jin et al. [31] proposed a novel architecture which collects data, store them in a database and display them on a real-time manner using a C#.net software. However, this obliges users to install this software on their computers.

Paper [32] proposes a wireless Internet-based observatory (ReCON) which goal is to develop a system to collect measurements from environmental sensors, and to store them in a buoy and a shore station node until the control center Linux computer transfers data

to an archiving system. Note that, authors have not specified how events are triggered to enable real-time delivery of measurements to the web displays.

2.1.4 Operating systems

Sensor nodes are different from traditional systems and networks as explained in Sec. 2.1.2. Some Operating Systems (OSs) have been designed to be tailored these characteristics. The OS runs on a single node and is responsible for managing resources and tasks on this node [33]. Moreover, it gives an abstraction level of the hardware for the programmer by providing common libraries and system services. Many OSs for WSNs [34] are available such as TinyOS [35], Contiki [36], Mantis [37], Nano-RK [38] and LiteOS [39]. There are some characteristics that make differences between them [34] :

- ***Architecture :*** it can be either (i) *monolithic* where the system is a collection of inter-connected services through interfaces, (ii) based on a *micro-kernel* which implements the minimum functionalities, and most of OS services are provided via servers, (iii) based on a *virtual machine*, or (iv) a *layered architecture*.
- ***Programming model :*** it can be either an event-driven programming or multi-threaded programming model.
- ***Scheduling and real-time support:*** scheduling determines the order in which tasks are executed. A real-time scheduling algorithm meets the deadlines of hard real-time tasks where a deadline is a given time after a triggering event, by which a response has to be completed.
- ***Memory management and protection :*** this is the used strategy, which can be static or dynamic, to allocate and de-allocate memory for different processes and threads.
- ***Communication protocol support :*** this refers to both inter-processes and nodes communication.
- ***Resource sharing :*** this refers to the orchestration of resource access between concurrent processes.

The most used OS in WSNs is TinyOS which is an open-source operating system for wireless sensor networks, featuring a component-oriented architecture. In addition, it is monolithic so it minimizes the code size as required by the severe memory constraints inherent to sensor networks. TinyOS provides developers different libraries like network protocols, distributed services, sensor drivers, and data acquisition tools – all of which can be used as-is or be further refined for a custom application [12]. Primarily, it was

event driven. Then, a support for threads has been added [40]. It uses a simple First In First Out (FIFO) scheduler. TinyOS and programs running on it are written using *nesC* which adds support to components, events handling and tasks to the C programming language [41].

2.1.5 Data management challenges

The main task of a sensor node is to monitor one or more physical phenomena, such as temperature, humidity, pression, chemical concentrations, etc. The observation process or data acquisition process consists in converting a physical phenomenon into a measurable electrical signal, then to a digital value using a sensor (e.g., a transducer) and a analog/digital converter. This process is done periodically by one or more sensor nodes in the network for one or more physical phenomena. Consequently, a huge amount of spatio-temporal sensor data is generated that needs to be well managed in order to realize the application objectives.

These data need to be routed between sensor nodes and/or transferred to a sink. Many routing approaches dealing with this problem can be found in the literature. In addition, these sensor data need to be processed to deduce conclusions and extract intelligence. There are three possible processing levels. Moreover, sensor nodes need to communicate with other networks like the Internet for the purpose of giving more services to users. Moreover, sensor data may be private and secret so that it is necessary to protect them.

2.1.5.1 Routing protocols

Sensor nodes communicate wirelessly using a routing protocol which defines the process of moving packets across the network from one node to another. The chosen protocol should take into account the network limitations, as shown in Sec. 2.1.2. It should provide scalability, reliability, low overhead, etc. They can be classified according to the techniques used throughout their execution. More fine grained classifications are proposed in [9, 42, 43]. They can be classified into six categories:

- ***flat-based routing*** where all sensor nodes play the same role.
- ***hierarchical-based routing*** where nodes play different roles as opposed to*flat-based routing*. The network is divided into clusters headed by a cluster-head which has more tasks to do such as data aggregation and fusion. This role can rotate periodically between cluster members. LEACH (Low-energy adaptive clustering hierarchy) [44] is the most known protocol belonging to this category.
- ***data-centric routing*** where there is no addressing mechanism and all communications are performed neighbor-to-neighbor. When source sensors send their

data to the sink, intermediate sensors can perform a form of aggregation on the data originating from multiple sources and send the aggregated result toward the sink [9]. Some protocols belonging to this category are SPIN [45], Rumor [46], and Cougar [47].

- ***location-based routing*** where the information position is used to compute the distance between two particular nodes for the purpose of estimating for example the energy consumption. Moreover, packets are relayed to the desired regions rather than the whole network. Some protocols belonging to this category are Geographic Adaptive Fidelity (GAF) [48], Geographic and Energy-Aware Routing (GEAR) [49] and Trajectory-Based Forwarding (TBF) [50].

- ***Multipath-based routing*** is different from *single-path routing*. In a single-path routing, when considering data transmission between a source node and the sink, the former selects the shortest path to send the message to the latter. In *Multipath-based routing*, the source node selects the first k shortest paths to the sink and divides its load uniformly among these paths. [51, 52] present two routing protocols belonging to this category.

- ***QoS-based routing*** provides a protocol that helps finding a balance between energy consumption and QoS requirements such as delay, reliability and fault tolerance. The SPEED [53] protocol belongs to this category.

2.1.5.2 Network convergence

Generally, nodes in WSNs communicate using a Wireless Personal Area Network (WPAN) protocol such as IEEE 802.15.4 [2] or Bluetooth [54] standard which makes them disconnected from a global Wide Area Network (WAN) such as the Internet. In order to handle this issue, multiple solutions have been proposed in the literature. This convergence promotes the concept of an Internet of Things (IoT) where sensor nodes (e.g., the things) are uniquely identifiable objects connected to the Internet. Some proposals aim at realizing the convergence between Mobile Cellular Networks and WSNs [55] or at interfacing WSNs to the core network using passive optical networks [56]. IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) is an advanced solution to adapt the Internet Protocol version 6 (IPv6) to sensor nodes and thus provide the convergence between WSNs and traditional IP networks [57, 58, 59, 60, 61, 62]. The proposed architecture of the network is presented in Fig. 2.3. Mapping from the IPv6 network to the IEEE 802.15.4 network poses some challenges [58]:

- ***Packet size adaptation :*** The maximum packet size is 127 bytes in the IEEE 802.15.4 standard, after the deduction of a maximum frame overhead introduced at the media access control and link layers of 25 bytes without security or

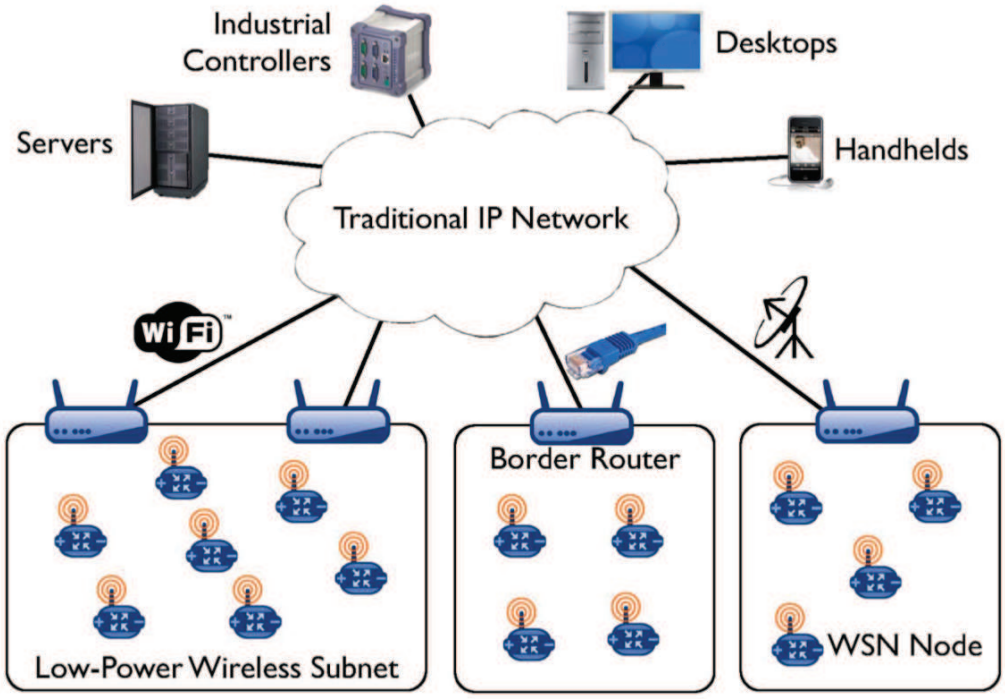


Figure 2.3: 6LoWPAN network architecture [57].

46 bytes after security application. It remains only 102 bytes or 81 bytes respectively which do not respect the IPv6 minimum value (e.g. 1280 bytes) for the maximum transmission unit (MTU) in IPv6 specification [62]. This issue was resolved using the techniques defined in the next points.

- ***Fragmentation :*** A new adaptation layer is added between the network layer and the data link layer. It receives the IPv6 network layer packets of 1280 bytes and sends them to their counterpart on the remote node using 802.15.4 frames. The payload size of these frames is only 81 bytes, the adaptation layer must fragment IPv6 packets before sending them and reassemble them at reception [62].

- ***Header compression*** mechanisms were first standardized in [62], then updated in RFC 6282 [61]. They are stateless mechanisms that remove redundant information accross the link, network and transport layers. Moreover, common values for header fields are assumed and compact forms of those values has been defined.

- ***Address compression and resolution :*** Sensor nodes may either use IEEE 64-bit extended addresses or, after an association event, unique 16-bit addresses within the Personal Area Network (PAN). IEEE 64-bit addresses are derived from link-layer addresses. All sensor nodes within the sensor network subnet share the same subnet ID [62] which is ignored in the compressed header (traffic class and flow label are 0, hop limit can be 1, 64, or 255, etc.).

- ***Network management mechanisms :*** The SNMP [63] framework widely used in traditional IP networks was optimized to fit 6LoWPAN requirements in [64].
- ***Routing*** is a two-phase problem that is being considered for 6LoWPAN networking, first mesh routing in the PAN space, and second the routability of packets between the traditional IPv6 networks and the 6LoWPAN network [62].
- ***Device and service discovery :*** The neighbor discovery protocol [65] was adapted for 6LoWPAN in [66].
- ***Security*** remains an open issue in 6LoWPAN networks [62].

REpresentational State Transfer (REST) is an architectural style for distributed systems and the World Wide Web (WWW) represents a conforming implementation to this style. Systems following the REST constraints are often referred to as *RESTful*. REST consists of client-server system where the former initiates requests using Uniform Resource Identifiers (URIs) and methods (e.g. GET, POST, PUT, DELETE, etc.) to the latter that processes them separately without saving any client context. Then, it returns appropriate responses which are resource representations. The interface is uniform between both sides for the purpose of enabling each part to evolve independently. Interface uniformity and stateless properties have encouraged the development of this service in resource constrained devices.

Thus, some works are providing RESTful services for WSNs such as TinyREST [67] and CoAP [68]. The Constrained Application Protocol (CoAP) is a specialized web transfer protocol to be used with constrained nodes and constrained networks and has been designed for machine-to-machine (M2M) applications. It includes the key concepts of the Web such as URIs and Internet media types. It implements four HyperText Transfer Protocol (HTTP) methods: GET, POST, PUT and DELETE. An implementation of this protocol is provided for TinyOS and Contiki. In [69], authors present a Binary Web Service (BWS) implementation for WSNs using the RESTful approach and the binary encoded XML. They show the opportunities that the 6LowPAN and web services give to WSNs in a real world system. The approach found in [70] integrates simple mashups and RESTful API in sensors and proxy gateways, which gives a new way to integrate sensor networks in the Internet of things.

The use of 6LoWPAN and REST web services pave the way for the integaration of WSNs in the Internet. Clients can consider sensor nodes as web servers and communicate with them through the Internet or sensor nodes can send them data to an external server using REST services. Whereas, sensor nodes does not have enough resources to handle/provide HTML and XML documents, and they are not able to work as servers as they have periodical sleep intervals most of the time. The opportunity of sending raw sensor data to an external server using a RESTful web service is considered in Chap. 3.

2.2 Data management

As mentioned before, data in WSNs is very important and its efficient management is a crucial need. In the following, the term data refers to "sensor data". There are three levels of data management in WSNs : *Data acquisition*, *Data Processing* and *Data Storage*. These features are discussed below, then two data management systems are presented.

2.2.1 Data acquisition

This defines how required data is delivered to applications. It can be either *event-based* or *query-based*.

The **event-based** pattern promotes the production, the detection and the consumption of events. When an application specifies its interest in some state changes of the data, it receives event notifications when changes occur. MIRES [71] and TinyDB [72] provides this feature. MIRES implements a publish/subscribe server in each node, as defined in Sec. 3.2.2. Sensor nodes start by advertising their topics, e.g., their measured data like temperature, in the network. A user application, connected to the sink, can subscribe to the desired advertised topics on sending subscription messages to source nodes. Then, sensor nodes publish the collected data associated with the subscribed topics.

In the **query-based** data model, the application considers the sensor network as a distributed database. The two best-known solutions for this approach are Cougar [47] and tinyDB [72]. TinyDB is a query processing system for extracting information from a network of sensor nodes. It provides a SQL-like interface to specify the data one want to extract, along with additional parameters, like the rate at which data should be refreshed. Given a query specifying data interests, TinyDB collects, filters and aggregates the data, and routes them out to a PC connected to the sink. Considering a user wishing to monitor the temperature of the sixth floor rooms of a building choose to do this using a tinyDB query that reports all rooms where the average temperature is over a specified threshold every 30 seconds. The query is illustrated in Listing 2.1.

```
SELECT AVG(temperature),room FROM sensors
 WHERE floor = 6
 GROUP BY room
 HAVING AVG(temperature) > threshold
 SAMPLE PERIOD 30s
```

Listing 2.1: Alert message schema.

Hence, TinyDB belongs to both categories, event-based and query-based. The feature, `SAMPLE PERIOD`, allows the definition of alerts, such as in Listing 2.1. The devices must report their data once per thirty seconds.

When the application wants to keep periodic track of data, it can subscribe to periodic events. Event has less downlink as the subscriber needs to subscribe once to receive

periodic updates. However, queries pays the round-trip penalty for each data query. In contrast, when handling sporadic data needs, alerting loses its edge due to the fact of sampling and transmitting not needed data. Thus, the choice between them depend on applications and situations. A compromise between these two categories can be selected and balanced according to the situation [73].

2.2.2 Data processing

Data processing is the fact of analyzing, aggregating, computing statistical results (average, maximum, etc.), transforming, translating, etc. data. In WSNs, this can be performed in the node providing the data, in a centralized processing unit where data is collected from the WSN, then processed in a centralized manner, or in the network using a distributed manner.

The second approach is very suitable to reduce data traffic in the network as it is reduced hop by hop. It can be used in query-based or event-based systems. For example MIRES, TinyDB and Cougar provide this feature, whereas on processing data, some information are lost and may be not suitable for data traceability. Thus, in certain surveillance applications, all sensor data must be kept in raw format for the purpose of analysing it when further analyzes are needed or a disaster occurs. Indeed, centralized processing was selected in many monitoring projects such as [74, 30, 27, 31, 29].

2.2.3 Data storage

Data storage defines where data is stored in the WSN. It can be done at three different levels. First, data can be stored in an external storage system such as a database in the base station [30, 27] or a GRID [74, 29]. Second, it can be stored locally where it was generated. Third, a tradeoff between the two approaches can be selected.

Storing data locally or in a hybrid mode is impossible for long term WSNs due to memory limitations on sensor nodes, node failures and the huge amount of collected data. Moreover, for better traceability, data should be stored for long periods in a reliable external storage system.

This categorization serves to better differentiate data management levels in order to understand each system characteristics and thus choose the adequate one for the application. In order to manage and keep data coming from a multitude of heterogeneous WSNs for long term, external data management systems are the most adequate.

2.2.4 External data management systems

Different data management levels were discussed above with a brief description of some systems such as tinyDB and MIRES. In this section, a description of two other systems

is provided.

2.2.4.1 QuadraSpace

The QuadraSpace Protocol [75] is an open-sensor network service for managing motes and collecting their data. The protocol defines a common way for sensor registration, data collection, remote communication and activation. The QuadraSpace architecture is composed of four main components: QuadraServer, QuadraWeb, QuadraGate and QuadraBox as depicted in Fig. 2.4. Communication between those components is guaranteed by the QuadraSpace protocol which also assures the communication between QuadraServer and QuadraSpace enabled sensors. QuadraSpace protocol is based on RESTful HTTP/HTTPS and XML, and it offers developer libraries for mobile and embedded platforms.

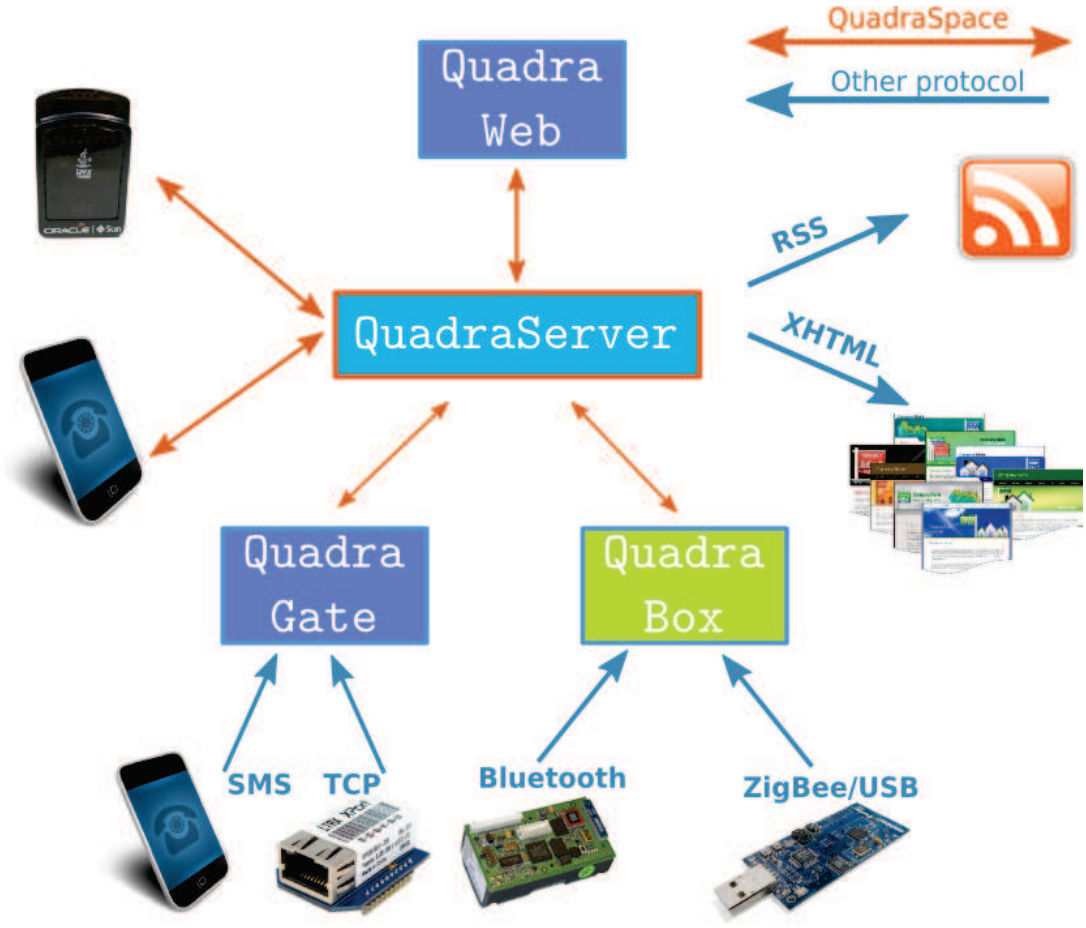


Figure 2.4: The QuadraSpace architecture.

The QuadraServer is responsible for the management of user accounts, mote (i.e., sensor node) registration, event-data collection and activating triggers on user-defined event conditions.

The QuadraWeb offers the possibility to export, analyze events and subscribe to user's events RSS feeds, in addition to web interfaces for user registration, mote registration, event management and trigger configuration.

The QuadraGate is the module responsible for accessing the QuadraServer using public networks and protocols other than HTTP REST. After receiving events notifications via SMS, MMS, email, etc., it encapsulates them in QuadraSpace events and sends them to the QuadraServer.

The QuadraBox works as a gateway between private local networks and the QuadraServer.

They are three XML objects in QuadraSpace protocol: mote, event and trigger. Each user creates an account for his objects. He registers his motes, which are identified by an ID and characterized by a manufacturer, a location, one or more sensors and one or more communication interfaces. Fig. 2.5 describes the structure of any XML mote object. An event is a data sample of a sensor or a heartbeat notified by a mote to the QuadraServer. An event may trigger an action. It consists in a type, a source, a location and a data payload, when type is 'sample' and contains the sensor reading value. Triggers are used to automatically execute actions when one or more conditions on incoming events are satisfied. An event object consists of an ID, a type, one or more conditions, and one or more actions.

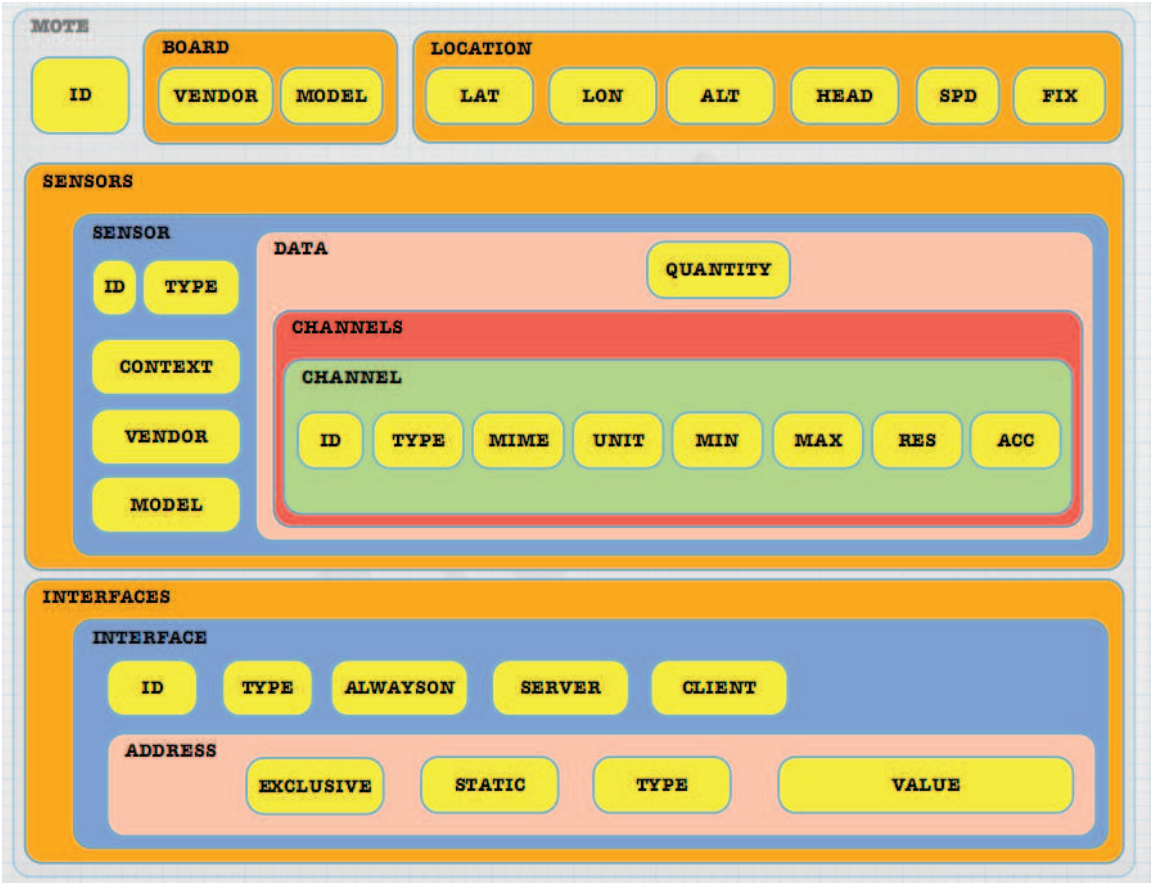


Figure 2.5: Structure of a mote object [75].

2.2.4.2 SWE: Sensor Web Enablement

The Sensor Web Enablement (SWE) project [76] aims at enabling interoperability between traditionally disparate community sensor networks and the realization of sensor webs. It is a suite of standards from OGC (Open Geospatial Consortium): three standard XML encodings (SensorML, O&M, TML) and four standard web interfaces (SOS, SAS, SPS, WNS).

Sensor Model Language (SensorML) : standard models and XML Schema for describing sensors systems and processes; it provides information needed for the discovery and the location of sensor observations.

Observations and Measurements Schema (O&M) : standard models and XML Schema for encoding observations and measurements from a sensor, for both archived and real-time ones.

Transducer Model Language (TransducerML) : conceptual approach and XML Schema for supporting real-time streaming of data to and from sensor systems.

Sensor Observations Service (SOS) : standard web service interface for requesting, filtering, and retrieving observations and sensor system information.

Sensor Planning Service (SPS) : standard web service interface for requesting user-driven acquisitions and observations, to (re-)calibrate a sensor or to task a sensor network.

Sensor Alert Service (SAS) : standard web service interface for publishing and subscribing to alerts from sensors.

Web Notification Services (WNS) : standard web service interface for asynchronous delivery of messages or alerts from any other web service.

Sensors, that are registered at a SOS, publish observation results to the service. The description of sensors in sensorML and SOS are registered in a catalog service (CAT) as described in the Fig. 2.6. The required steps by the user to obtain the needed observation data are:

1. the user sends a *search request* to the catalog.
2. the catalog answers with a list of SOS service instances that fulfill the requirements.
3. the user binds the SOS and retrieves the observation data, encoded in O&M.

If the catalog does not provide any SOS instance that fulfills the requirements of the request, the user can search for a link to an SPS instance in the catalog, and assign a

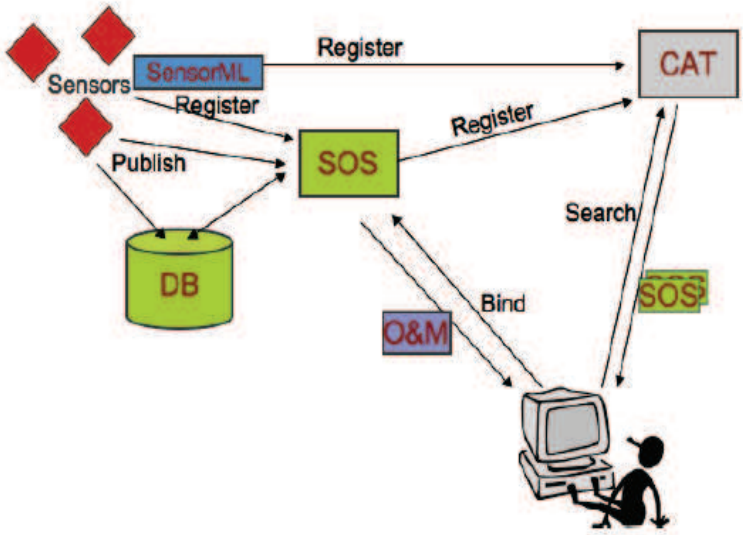


Figure 2.6: SWE standards interactions -1- [76].

task to sensors through the SPS to produce the desired data. Then, SPS informs the user about data availability either directly or through the WNS if the tasking is delayed. Those steps are described in Fig. 2.7.

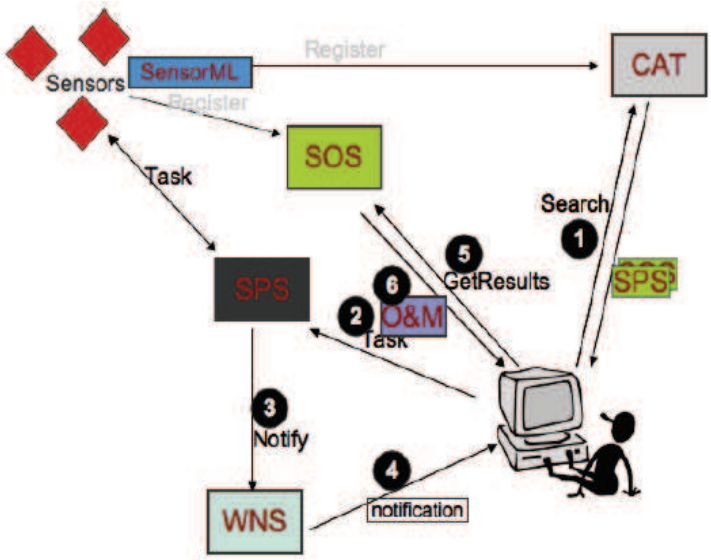


Figure 2.7: SWE standards interactions -2- [76].

Another use case is when the user wants to get notified immediately when a condition is triggered. Once again, the client receives information about appropriate SAS from a catalog and subscribes to the SAS. Sensors publish observation results continuously to the SAS. The SAS handles all the filtering and alerts the client if the subscription condition is matched. The SAS either sends the alert directly to the client, or makes use of the WNS in order to deliver the alert message. Those steps are described in Fig. 2.8.

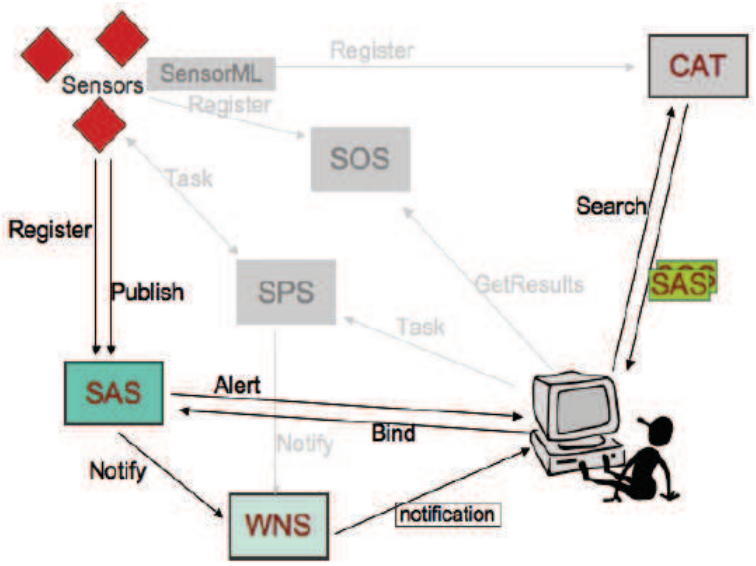


Figure 2.8: SWE standards interactions -3- [76].

2.2.4.3 Sensor Bean

A new approach, called Sensor Bean, has been proposed based on a service-oriented component model [77]. Sensor Bean (see Fig. 2.9) is a component with three pairs of input/output interfaces. The first one, **service requester/provider**, is a port for synchronous method calls to request/publish services. The second one, **event sink/source**, is used to subscribe/publish events. Finally, the third one **produces/consumes** sensor measurements.

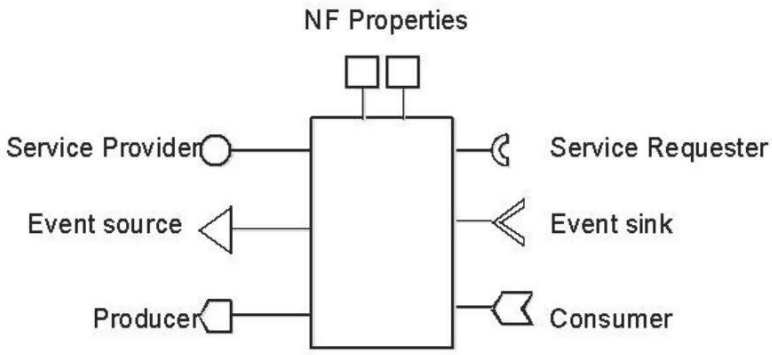


Figure 2.9: The Sensor Bean component model [77]

An application in the Sensor Bean approach consists in two layers. The first layer is composed of one or several gateway(s) built on top of an OSGi framework and using sensor beans. Those beans are connected to sensors to collect data. The second layer is a J2EE server built using web pages and sensor beans, and connected to gateways. In addition to visualization, it ensures the storage of sensor data in a sensor warehouse database as described in Fig. 2.10. SensorBean provides a description of a component-

based architecture for sensor data collection without providing a mechanism to facilitate component reconfiguration.

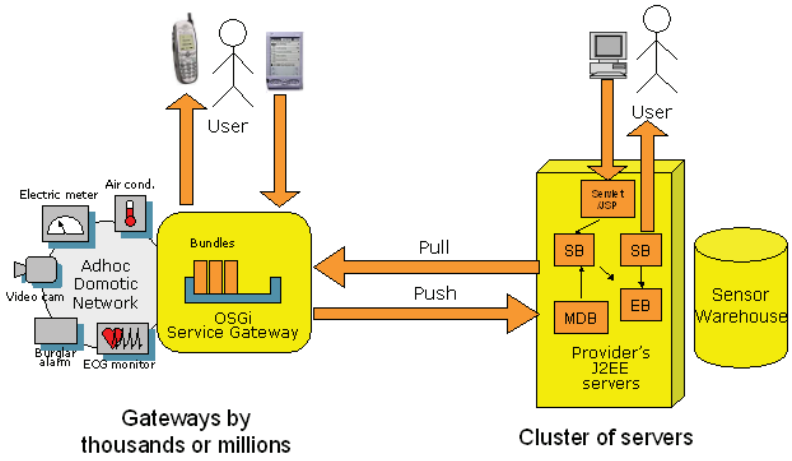


Figure 2.10: Distributed architecture for gathering sensor data [77]

Conclusion

These projects support data collection from heterogeneous sensor networks. However, none of them provide a complete implementation, and the real-time issue has not been a priority concern. The QuadraSpace project does not provide any specifications on how to store, index and extract data from the middleware. The work done in the SWE project is adapted to complex sensors and thus provides complex XML schemas, not suitable for sensors with limited memory and storage capabilities. None of the projects have a mechanism to be integrated with deployed WSN and well-known sensor operating systems, like TinyOs and Contiki.

2.3 Cloud and WSNs

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts remote services with user's data, software and computation [78]. Cloud computing comes into focus when one thinks about increasing capacity, adding capabilities, having reliable shared services and servers, having high-capacity networks, low-cost computers and storage devices, acquiring or delivering a configured resource on demand. The key technology of cloud computing is virtualization which is the ability to separate OS and applications from the hardware. A Virtual Machine (VM) is a tightly isolated software container that can run its own OS and applications as

if it were a physical computer. As a result, many VMs or virtual servers can be hosted on a single physical computer.

The use of these techniques in WSNs can be very beneficial in terms of improving their flexibility, re-usability, scalability and programmability and reduce their cost. Virtualization can be used to virtualize sensor nodes and/or networks while the cloud can be used to store the collected data from the sensor network or to share Virtual Sensor Networks (VSNs).

2.3.1 Sensor data in the cloud

The amount of data generated by WSNs may be huge, so storing it in the cloud, which offers long term and low-cost storage service, was investigated in many works. For example, in [79], authors present a home healthcare system in the cloud which offers to stakeholder, e.g., patients, family members, pharmacists, etc, three services: drug therapies management, sleep and light management and physical acticity management. Then, two strategies are followed to identify the security and privacy challenges of the application: the business logic strategy and the architecture-driven strategy to the end that patients be able to control the distribution and use of their personal data. In fact, the proposed security scheme is perfomed using two steps. First, patient's data are encrypted using a symmetric key before being uploaded to the cloud. Second, the key is distributed to authorized parties of decryption using either Attribute-Based encryption schemes or licences. To sum up, authors present the architecture of the system and the used security scheme to protect patient's personal data. Data storage in the cloud facilitates data sharing between actors but the security scheme limits this opportunity because of the difficulty of managing and distributing keys. This issue is investigated in Chap. 6.

Authors in [73] present a three-tiered architecture named the Cloud-Edge-Beneath Architecture in which sensors (Beneath layer) are connected to the cloud indirectly through an edge computer (Edge layer). The acquisition of updated sensor readings can be realized via communications between cloud, edge, and beneath layers using information push and pull mechanisms which are respectiveley event-based and query-based mechanisms, as explained in Sec. 2.2.1.

As push has less downlink than pulling, authors propose a hybrid approach to achieve a near-optimal energy cost to balance the tradeoffs between pushing and pulling. The optimisation is ideal for helping applications to switch between the two modes, but some applications require periodic measurements, even if there is not a disaster or emergency.

2.3.2 Virtualization and WSNs

Some virtual machine based middleware have been proposed to run on sensor nodes, such as Maté [80] and MagnetOS [81]. They allow to reprogram the network after nodes

deployment. Melete [82], a based system on Maté, enables the execution of concurrent applications on a single sensor node. In addition, it provides dynamic grouping to offer flexible and on-the-fly deployment of applications based on contemporary status of the sensor nodes. This feature allows program updates in already deployed sensor nodes in the field but the issue remains the limited memory size of Sensor Nodes (SNs) and consequently the size and number of VMs.

After the advances in the virtualization of *sensor nodes*, the virtualization of *sensor networks* was studied in many works [83, 84] and is the subject of many ongoing projects [85, 86].

In [83], authors presents a multi-layer task model for Body Sensor Networks (BSNs) based on the concept of Virtual Sensors to improve architecture modularity and reusability. Every processing task can be represented as a virtual sensor. Thus, the complete BSN processing part can be modeled as multi-level hierarchy of virtual sensors. The solution is composed of three building parts: Virtual Sensors (VS), Virtual Sensor Manager (VSM) and Buffer Manager (BM). When a user requests certain outputs given specified inputs, the VSM handles this request and configures a set of VSs to handle the computation task and connects them. VSs use the BM to setup communication through the use of buffers. Once configured the system is activated, and virtual sensors cooperate to produce the final outputs. A table, that maps each available combination of possible inputs and outputs to the appropriate VS implementation, is managed by the VSM and can be (re-)configured at run time. The implementation of this solution relies on SPINE2 [87] which is a framework for the development of signal processing applications on WSNs through a task oriented programming abstraction.

2.3.3 Sensor-Cloud Infrastructure

In [84], authors propose a Sensor-Cloud Infrastructure that provides a sensor system management to virtualize a physical sensor as a virtual sensor on the cloud computing and to let users request the use of virtual sensors or virtual sensor groups that satisfy their requirements. Dynamically grouped virtual sensors are provisioned automatically in response to user requests. In addition, it provides a portal as a user interface for registering or deleting physical sensors, for requesting, provisioning or destroying virtual sensors, for controlling and monitoring virtual sensors, and for registering or deleting users. The most important advantage of this solution is that users need not worry about the real location and the differences of multiple physical sensors. However, in the majority of sensor applications, the location of physical sensors is very important, and hiding this information is not suitable for those applications.

A business model of the virtualization of sensor networks was proposed in [88] where two types of providers collaborate to offer services to the application level users (ALU).

The first provider is the sensor infrastructure provider (SInP) that manages the physical sensor infrastructure. The second is the sensor virtualization network service provider (SVNSP) that hires resources from one or more SInPs to form VSNs, deploys customized protocol and offers services to ALU. This is illustrated in Fig. 2.11. This model differs from the one defined in Chap. 3 and 6 where each user or organization manages its own sensor data and they are free to share or sell it using the alerts, the RESTful API or the social portal. However, this model enables infrastructure sharing which is not useful in certain applications like healthcare monitoring.

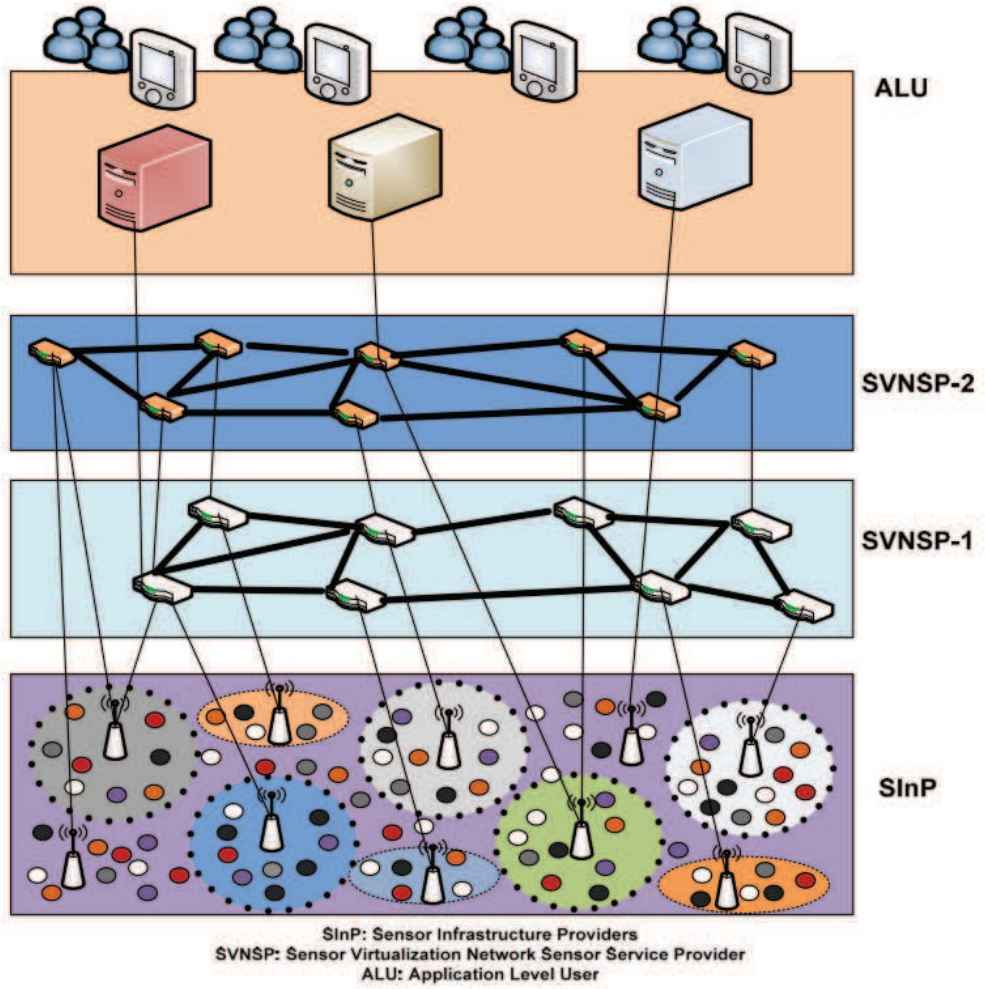


Figure 2.11: Business model of sensor network virtualization [88].

In summary, the combination of virtualization, cloud computing and WSNs gives new opportunities, applications, services and challenges. The virtualization of sensor networks and their reprogrammability, which is the ability of reprogramming sensor nodes and networks after deployment, are important opportunities. But the usefulness and the impact of deploying VSNs on security and overhead communication traffic was not well studied.

2.4 Sensor networks and social networks

Since their introduction, Social Network Sites (SNSs) such as MySpace, Facebook, twitter and googleplus have attracted millions of users, many of whom have integrated these sites into their daily lives [89]. For example, on october 2012, Facebook has announced a historic milestone of one billion active users. SNSs allow each user to have a profile, construct a list of other users with whom they share a connection (*friends*), share contents, and see the shared contents of their friends.

A web portal is a web site, built thanks to social mashup engines that brings information from diverse sources in a unified way. Usually, each information source is represented by a gadget and is displayed in a dedicated area on the page.

WSNs can use existing SNSs for sharing sensor data or built their own SNSs. This combination is very important as it enables the collaboration between *friends* and the sharing of sensor data.

In [90], authors propose to use existing social networking infrastructures and their web-based APIs in order to integrate Smart Homes to the web, offering social status to physical devices. Thus, the data is only published in SNSs which limits the opportunities offered by these sites.

In [91], authors present a framework, named SenseFace, that creates an overlay network to deliver the captured sensor data from a Body Sensor Network (BSN) to one's community of interest (COI). SenseFace can be seen as a four-tier application. The first tier, BSN, consists of several sensors that collects sensor data and sends it on a real-time manner to the second tier. The second tier is a personal gateway, which can be a mobile phone. It locally checks the incoming sensory data for possible threshold crossing, if any such violation of normal data is found, latest data and user ID will be sent to the next tier. The third tier is responsible of receiving, processing, storing and delivering appropriate sensor data to the overlay network, i.e. the fourth tier. Three components are involved. First, the *sensor data receiver* receives data from the personal gateway, semantically sort the sensor data from user information and store them. Second, the *COI manager* uses a social network analysis (SNA) process in order to maintain one COI intelligently. The SNA process [92] collects one's social network ties from different existing sources e.g. personal website, blog, email, CV, local computer folders and other relevant sources. This can also be provided manually by the user. Then, it categorizes the one's relations to some high level concepts, which are called personal social networks, such as friends, kin, colleague, medical, etc. Then, an overlay network, called COI, is created on top of the personal social network layer where each subgroup of the COI is mapped with a sensory data types. Techniques like fuzzy ontology and swarm intelligence are used during this process. Third, the *content adaptation* adapts the content based on end user devices and the social network the data are published on. Regarding the fourth tier, SenseFace comes

with a service to send messages to on-line popular social networks and other technologies, e.g. SMS, MMS and fax, in a real-time manner. In [93], authors present an application of SenseFace to the e-health domain. The automatic generation of social ties is a novel technique but to what extent it is reliable to support the management of critical data such as healthcare monitoring data.

In [94], authors present Sensorpedia which is a Web site that creates a human-machine network to connect sensors with users and applications for enabling global-scale sensor information sharing. Sensorpedia system is divided into two main parts. First, a Web based application is used to explore, contribute to, and share data from online sensor systems. The main user interface of the application is a map-based mashup. Second, the application programming interface (API) provides the software instructions for interfacing sensors, sensor data, and sensor alerts. In addition, Sensorpedia allows users to establish trusted groups, share relevant views and sensor data, and publish updates to external applications such as Facebook, and Twitter. Sensorpedia's primary activity is sensor data sharing. Its social objects are the sensors and observation data. Its feature set provides the functionalities: register sensors, establish social networks of trusted collaborators, and explore available sensor data.

Sensor data formats are heterogeneous, so they need a specific component to adapt and display them using an adequate visualization feature. Moreover, this convergence between social sites and sensor networks raises new privacy challenges. These points were not well addressed in theses works and are more discussed in Chap. 6.

2.5 Conclusion

This chapter introduced WSNs, their applications and data management challenges. Then, different levels of data management in these networks have been discussed and a survey of some existing solutions. Finally, opportunities of convergence between WSNs and both cloud computing and social networking were presented along with some related works in literature.

Limitations of WSNs and data management issue are tackled in Chap. 3 and 4, where a flexible and extensible middleware is proposed, then applied in the field of water quality monitoring.

Chapter 3

iSensors : A middleware for dynamic sensor networks

Contents

The presence of sensors in several mobile devices together with the development of WSN systems for several domains makes each person or organization possesses heterogeneous sensors. These sensors are not supposed to communicate with each other, and they generate a huge amount of data with different formats. The ability to manage these data, to extract useful information, and to generate alerts is a need facing many limitations and problems. These limitations are both hardware and software. Hardware limitations are due to the limited amount of resource available on a sensor node (processing, communication, storage, energy), while software limitations are due to the MAC protocol in WSNs

which allows nodes to have periodic sleep intervals to reduce their energy consumption that makes nodes periodically unreachable.

3.1 Problem statement

As presented above, each person or organization possesses several heterogeneous sensor nodes in disparate networks which generate a huge amount of data with different formats. Heterogeneity makes providing a common data management system a challenging task as it should be flexible and configurable. Moreover, these networks are mostly deployed and application specific which limits any attempt to update their programs or to load new ones. Therefore, the use of existing solutions in literature becomes impossible because if one needs to update sensor nodes programs like TinyDB [72] and Cougar [47], others require a protocol or a specific data format like SWE [76] and QuadraSpace [75].

Furthermore, the system collecting and managing the data from sensor nodes should take into account this heterogeneity and understand all inputs in order to be able to process them. Thus, it must be flexible in order to adapt itself to this heterogeneity. Therefore, data acquisition and processing in such a system must be externally to the network. So, this system can run on a data center which can be beneficial as it offers more storage space and computation capabilities.

In order to interconnect sensor networks to the system collecting the data, a communication standard must be used and it must be compatible with the majority of systems and programming languages. The HTTP standard fulfills these requirement and offer two types of web services: Representational State Transfer (REST) and Simple Object Access Protocol (SOAP). The first one is simpler and lighter as it is stateless and does not use a lot of extra xml markup or specific toolkits. Moreover, it is compatible with OSs of WSNs, as presented in Sec. 2.1.5.2, and most of the programming languages. Hence, it is advantageous to use RESTful services. Incompatible sensor networks can use a dedicated gateway to interconnects them to the system.

Moreover, preferably the system offers a private space for each physical node communicating with it. Also, there are some common functions to run on the system for heterogeneous sensors, like reception and storage services. So, it is important to reuse the code. In addition, such a system must hide the dynamicity of appearance and disappearance of nodes in the WSNs and enables adding and deleting nodes at runtime. For these reasons, the use of a component-based programming model is interesting.

In order to tackle with this problematic, this chapter proposes a new dynamic middleware managing the reception, the storage, the indexing of data and the generation of alerts from heterogeneous sensor nodes. In addition, the middleware is fully customizable and configurable [95, 96]. The remaining of this chapter provides in Sec. 3.2 a description of our proposed middleware architecture, followed by the implementation description of the

iSensors middleware in Sec. 3.3. Sec. 3.4 presents some possible middleware extensions. We wrap this chapter with a conclusion in Sec. 3.5.

3.2 Architecture

The iSensors middleware is made of a software installed on a server and accessible to physical nodes via the Internet or local network. Fig. 3.1 describes the global architecture of the middleware that is supposed to be hosted in an OSGi container. The communication model between the physical nodes and the middleware is based on the push function where the node pushes data periodically into the middleware using RESTful API. The push frequency must be configurable in the view of optimizing the generated communication traffic. This middleware is composed of two essential elements: a set of composites and an eventing server.

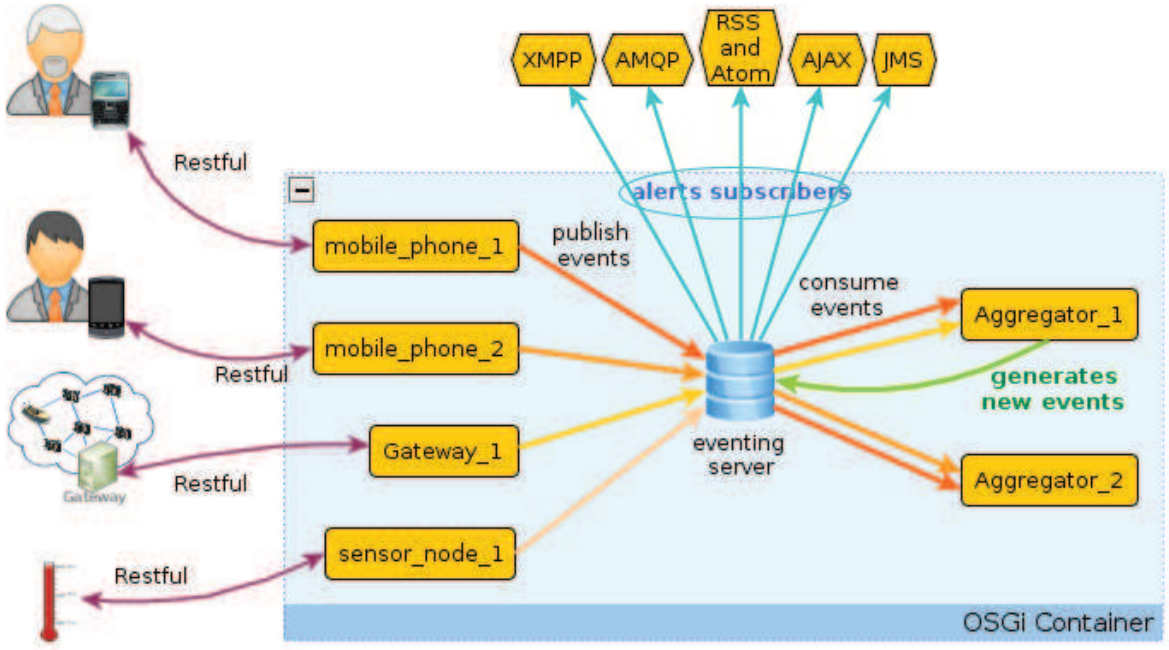


Figure 3.1: Global architecture of the iSensors middleware.

3.2.1 Composite

A composite is a software component that is connected to a physical node like a sensor node, a mobile phone or a gateway. It can also be used to aggregate and process data from different composites. Each physical node that has the capability to communicate with the middleware should be setup as a composite. There exists three families of composition:

1. *a sensor composite* is connected to a physical sensor node having one or more sensors.

2. *a gateway composite* is connected to a physical gateway node which interconnects a WSN, composed of one or more sensor nodes, and the middleware.

3. *an aggregator composite* has subscribed to different measurement sources in order to collect sensor data, then analyses and aggregates them. It can then publish or store the aggregated data and/or genenate reports.

The second family is used to hide the dynamic aspect of the network like a wireless sensor network, where sensor nodes appear and disappear dynamically. Otherwise, installing and removing nodes would be difficult for the middleware administrator. The generic structure of the first and second families is described in Fig. 3.2a. The difference between those two families is limited to the configuration of the RESTful API, while the generic structure of the *aggregator node* is described in Fig. 3.2b.

3.2.2 The eventing server

The eventing server is based on the publish-subscribe messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into topics, without knowledge of what, if any, subscribers there may be. Similarly, subscribers express interest in one or more topics, and only receive messages that are of interest, without knowledge of what, if any, publishers there are. In addition, it can route events from the middleware to external subscribed applications using many messaging protocols (RSS, JMS, XMPP, ajax, etc.). Therefore, composites can be publishers and/or subscribers. The eventing server is used to route messages to external subscribed applications and is meant to act as a communication mechanism between even heterogeneous composites. Hence, an aggregator composite subscribes to topics of interest in order to aggregate the published data.

3.2.3 Composite design

A composite might be understood as a super-component composed of other composites or components. It isolates its sub-parts to build a private space. It is identified by a family and a unique identifier.

In this middleware, each node is a composite that instantiates components and/or other composites, also called sub-components in the remaining part of this thesis. This makes the node a private unit that does not interfere with other nodes. Those sub-components are interconnected using services. So, a sub-component can provide or consume a service and its configuration is done using a service that it provides, or using the constructor at instantiation. Some configurations can also be done using the RESTful API to setup new or modify existing scripts and variables.

In order to easily add new nodes to the middleware, there are default components that produce default services. Thus, adding a new composite is limited to the instantiation of some default components.

Default (Inner) components

Fig. 3.2a and 3.2b describe the default structure of a composite. Important sub-components are:

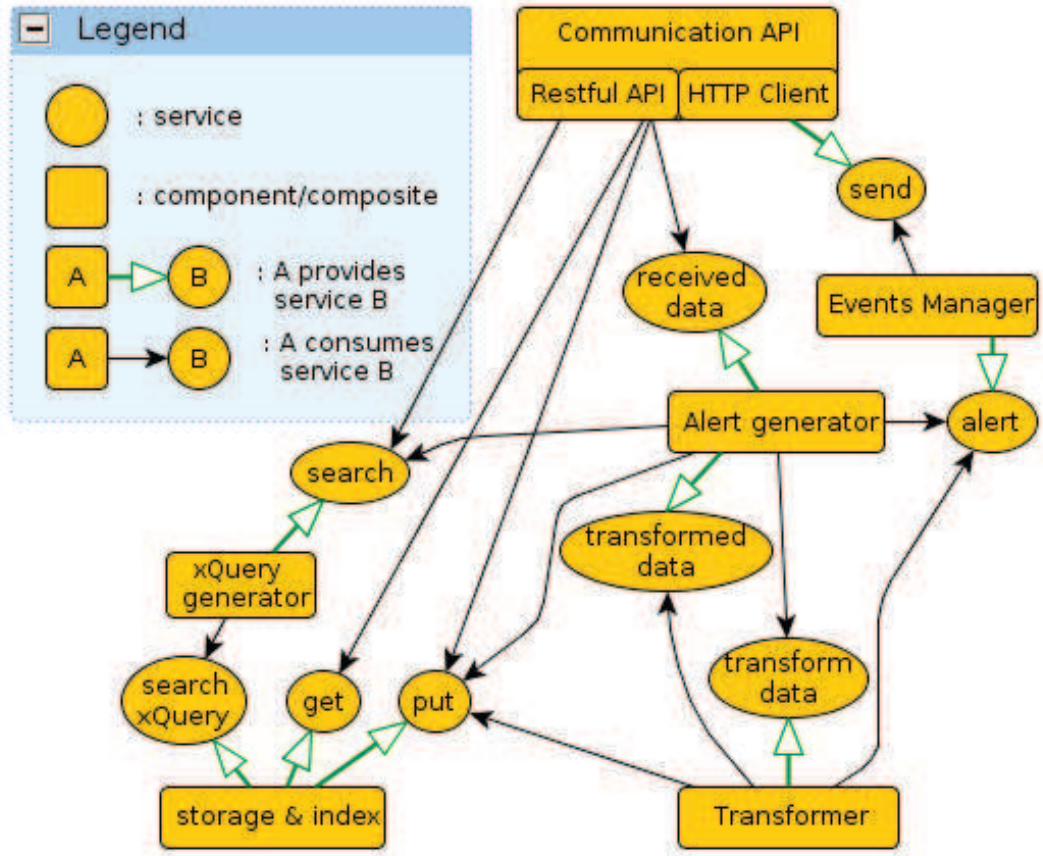
- **The Storage & Index component** is responsible for permanently storing and indexing data. It provides services to put, get and search for data. Put service returns a unique identifier of data which can be later used to get the data. In the middleware, an embedded component connected to existDB [97] is provided. It can also be provided by another party to be connected to a cloud storage and indexation solution for example.

- **The xQuery generator component** is responsible on producing an adaptation layer between the *storage & index component* and the other components using the search service of the former. This component translates a generic query (provided using the xQuery language) to allow the latter components to be independent from the used solution for storing and indexing data.

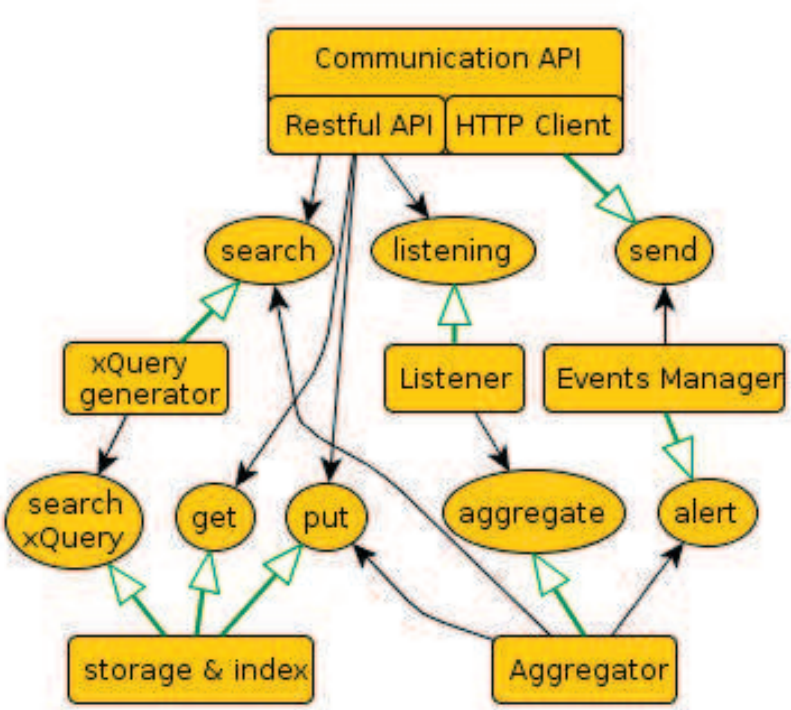
- **The alert generator component** is responsible for generating alerts by processing the received and/or transformed data. The generation is performed through the execution of a dedicated function on the input data. In order to allow the xQuery script to execute the reasoning operations, a version of the last data is injected in the script. In addition, it can be wired to the search service for more advanced scripts. At this level, the component generates an alert message and additional meta-data can be inserted to the input. This process is described in Fig. 3.3. The alert message schema is described in Listing 3.1.

- **The event manager** is responsible for routing events provided by the other sub-components to the best available server managing this type of event. For example, XMPP [98] and JMS [99] alerts are redirected to the *Eventing server*, while another server is needed to route SMS or email messages. As depicted in Listing 3.1, the type of the alert is given after parsing the alert XML file and extracting the value of the "type" tag.

- **The transformer** is used to translate and/or add meta-data information to data using an xQuery function. For example, it can be used to translate received sensor data in any format into the XML format or any other data format and at the same time add a data validation status after executing an analysis function having the input data and configuration settings as input parameters.



(a) A *sensor/gateway composite* structure.



(b) *Aggregator composite* structure.

Figure 3.2: Different composite structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="GeneratedAlerts">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="memory" minOccurs="1" maxOccurs="1" type="xs:string"/>
    <xs:element name ="alerts" type="AlertsList"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:complexType name="AlertsList">
  <xs:sequence>
   <xs:element name="alert" type="AlertType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:simpleType name="TypeOfTheAlert">
  <xs:restriction base="xs:string">
   <xs:enumeration value="jms"/>
   <xs:enumeration value="rest"/>
   <xs:enumeration value="rss+atom"/>
   <xs:enumeration value="xmpp"/>
   <xs:enumeration value="email"/>
   <xs:enumeration value="sms"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="Content-typeType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Text/xml"/>
   <xs:enumeration value="application/atom+xml"/>
   <xs:enumeration value="application/json"/>
   <xs:enumeration value="text/plain"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="AlertType">
  <xs:sequence>
   <xs:element name="type" type="TypeOfTheAlert"/>
   <xs:element name="content-type" type="Content-typeType"/>
   <xs:element name="destination" type="xs:string"/>
   <xs:element name="msg" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Listing 3.1: Alert message schema.

- **The communication API component** implements a RESTful API to configure and interact with composites. First, it receives data from the physical nodes. Second, it provides a service to send data (response, calibration, configuration, etc.) to the physical node. Third, it is the interface to navigate and search for the physical nodes' data. Finally, it gives the ability to configure the other sub-components, like updating the xQuery functions of the *alert generator* sub-component. The Application Programming Interface (API) specification of a sensor node composite is given in Table 3.1.

- **The aggregator** is a component of the *Aggregator composite* used to run a script on many data inputs to be aggregated based on parameter and configuration variables and the aggregate function defined in the script. Input data can be sensor data

coming from a single or many physical sensor nodes. In the second case, data are aggregated to be reduced to less output, e.g., the sensor data of many temperature sensors in a building are aggregated to extract the maximum value of temperature in order to detect a fire when this value exceeds some threshold. While, in the first case, input data can be issued from a single sensor node at different time slots. As an illustration, a sensor node can be immersed in water and capture several water quality parameters at different depths. Then, the role of the aggregate function can be the production of a report summarizing the measurement campaign.

- **The listener** is a component of the *Aggregator node*. It subscribes to topics in the Eventing Server that interests the aggregator sub-component. Once new events are published, they are caught and forwarded to the latter.

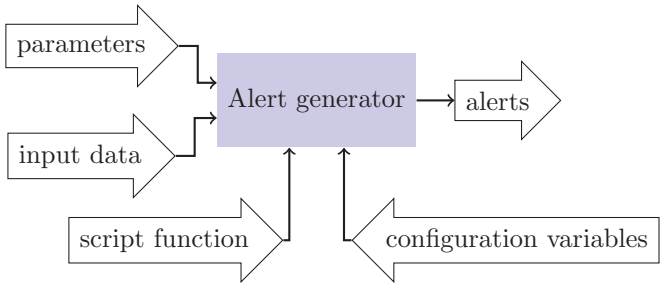


Figure 3.3: Alert generator flows.

Ressource isensors/nodes	POST (Create)	GET (Read)	PUT (Update)	DELETE
Easy API				
—/{nodeid}/	ERROR	Node description OR id	Update/create node description	ERROR
—/—/data	Add new data (message, packet, ...)	Last sent data	ERROR	ERROR
—/—/—/{data_id}	ERROR	Get the data	ERROR	ERROR
Advanced API				
—/{nodeid}	ERROR	{nodeid}, description, list of sensors and channels	Update/create node description	ERROR
—/—/sensors	Create new {sensor_id} and descriptions	List of sensor IDs	ERROR	ERROR
—/—/—/{sensor_id}	ERROR	{sensor_id}, description, list of channels	Update/create sensor description	ERROR
—/—/—/—/channels	Create new {channel_id} and descriptions	List of channel IDs	ERROR	ERROR
—/—/—/—/—/{channel_id}	ERROR	{channel_id}, description	Update/create channel description	ERROR
—/—/—/—/—/—/data	Add new data (message, packet, ...)	Last sent data	ERROR	ERROR
—/—/—/—/—/—/{data_id}	ERROR	Get the data	ERROR	ERROR
Configuration API				

... Continued on next page ...

... Continued from previous page ...

Ressource	POST (Create)	GET (Read)	PUT (Update)	DELETE
—/—/config	Create a new configuration file at the node_id local repository (HTML form to metalist)	ERROR	ERROR	ERROR
—/—/—/{config_id}	Create/update configuration file with {config_id} (HTML form to metalist)	GET the configuration file {config_id}	Update the configuration file {config_id} by uploading a new file	Delete the configuration file {config_id}
Script API				
—/—/script	Create new script file at the {node_id} local repository using an uploaded file	ERROR	ERROR	ERROR
—/—/—/{script_id}	ERROR	GET the script file {script_id}	Update the script file {script_id} by uploading a new file	Delete the script file {script_id}
Search API				

... Continued on next page ...

... Continued from previous page ...

Ressource	POST (Create)	GET (Read)	PUT (Update)	DELETE
—/—/search/[ldm][1-*]	ERROR	Get the result of the search query. The highlighted part in red is required and takes value l, d or m for last, day or month respectively. The highlighted part in green is optional and can take a value in $\mathbb{N}^*$. The default value is 1.★	ERROR	ERROR
—/—/—/sensors/{sensor_id}/[ldm][1-*]	ERROR	Same as in ★. Search only in {sensor_id} data.	ERROR	ERROR
—/—/—/—/—/channels/ {channel_id}/[ldm][1-*]	ERROR	Same as in ★. Search only in {channel_id} data.	ERROR	ERROR

Table 3.1: RESTful API specification.

END

3.2.4 Communication model

The default communication model between the middleware and the external entities is based on the RESTful API and the *Eventing server*, while the communication between components and composites is based on the *Eventing server* and service wiring.

When data or meta-data are received from a physical node by the *communication API*, they are stored, indexed, and sent to an alerting sub-component for analysis and alerts are generated if needed. Then, data are translated into another format such as SensorML or O&M. Afterwards, the translated data are again stored, indexed, and sent to the alert generator sub-component. This process is described in Fig. 3.4.

The event manager sub-component is responsible for routing generated events to the best server managing this type of events. Reasoning is applied in the alert generator sub-component by analyzing input data only or both input and history data.

The value returned to the sender (the physical node) must contain a unique identifier of the received data that can be used later to retrieve data using a GET query. In addition, it can contain a command or an update of a configuration parameter.

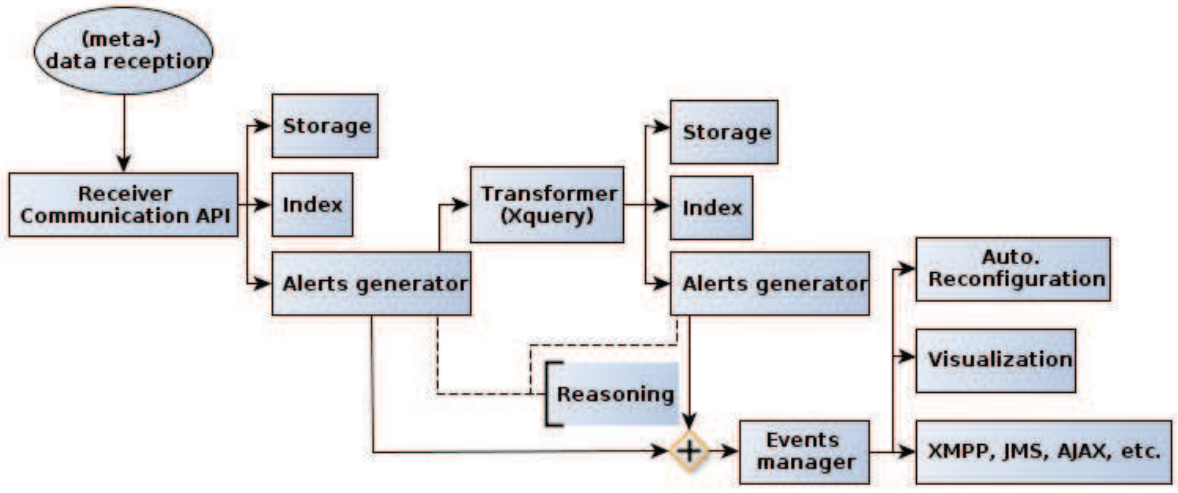


Figure 3.4: Communication steps after receiving data from the physical node.

3.3 Implementation

In order to implement the middleware while retaining the concept of composition and separation of component instances, the OSGi and iPojo technologies, described below, have been chosen. The key point of these technologies is also the possibility to reconfigure the structure, delete and deploy new components at runtime. This adds the dynamicity to the proposed iSensors middleware.

- **Open Service Gateway initiative** (OSGi) alliance [100] provides specifications, reference implementations, test suites and certification to develop lots of systems. The OSGi technology is a set of specifications that defines a dynamic component system. Developers can hide their implementations from other components while communicating through services, which are objects that are specifically shared between components. The *OSGi framework* is a module system and service platform for the Java programming language that allows to remotely install, start, stop, update, and uninstall bundles, which contain applications or components without requiring a reboot. The application life-cycle management (start, stop, install, etc.) is done via APIs. The framework run a service registry that allows bundles to detect the removal of services, or the addition of new services, and adapt accordingly.

- iPOJO [101] is a service-based composition extension of OSGi. It aims at separating functional code (the POJO which stands for Plain Old Java Object) from the non-functional code (i.e., dependency management, service provision, configuration, etc.). At run time, iPOJO combines the functional and non-functional aspects. It provides an Architecture Description Language to describe components and wires between them. It also supports the dynamic substitution and evolution of used services and components at runtime.

The default components are provided and installed in the karaf OSGi container [102] as iPOJO components. The composite is an iPOJO composite which is an instantiation of a set of installed components. The middleware is fully customizable as new sub-components and services can be added easily in the OSGi container. In fact, new composites and services can be installed and connected at runtime. The setup of a new one is done by uploading a bundle containing a meta-data file into the OSGi container. A simplified sample of a sensor composite setting file is given in Listing 3.2. This composite instantiates and connects some default sub-components.

3.4 Extensions

The middleware is based on composition and services which makes it extensible to provide other services apart from those described in Sec. 3.2. This section outlines some possible extensions of the middleware.

3.4.1 Sensors computation outsourcing

As presented above, sensor nodes have limited resources while the middleware should be hosted on a high-performance server which has more abundant resources of computation, storage and communication. Under those circumstances, it will be interesting to let

```
<ipojo>

<!--  Declares a composite -->
<composite name="sensor-1">
 <!--  Instantiates an instance of the communication API -->
 <instance component="eu.tsp.isensors.test.Node" >
  <property name="alias" value="/isensors/nodes/333-214"/>
  <property name="nodeID" value="333-214" />
 </instance>

 <!--  Instantiates an instance of the storage and indexation component -->
 <instance component="eu.tsp.isensors.impl.storage.StorageExistDBImpl" name="
     ExistDBservice" />

 <!--  Instantiates an instance of the Alert Geneator -->
 <instance component="eu.tsp.isensors.impl.alert.gen.AlertGenerator" name="
     alertGenService"/>

 <!--  Instantiates an instance of the Alert Geneator -->
 <instance component="eu.tsp.isensors.impl.alert.manager.AlertManager" name="alertManager
     "/>

 <!--  Instantiates an instance of the TRansformer -->
 <instance component="eu.tsp.isensors.impl.transformer.MessageTransformer" name="
     MessTrans"/>

</composite>

<!--  Instantiates an instance of our composition -->
<instance component="sensor-1"/>

</ipojo>
```

Listing 3.2: A sensor composite instantiation.

the user add a component/composite to provide a service that will be executed in the middleware. Therefore, the physical node sends the inputs to its composite which calls the service and returns the output to the node. A typical example is when a user having a smartphone, set as a composite in the middleware, wants to convert a recently captured video in order to send it in an email or to share it in a social site. For this to be done in the smartphone, conversion algorithms, calculation and memory resources are required which consume lots of energy. Therefore, calling a conversion service in the middleware reduces the mobile charge and offers a better experience for the user.

3.4.2 A market of Composites

As seen above, each user/organization has many sensor nodes which have no support for their data management, and due to their heterogeneity, it becomes more and more difficult to know their message types and how to communicate with them. Thus, providing a market for manufacturers and developers to offer specific components and composites for their products seems to be an interesting feature. Thereby, users could find specific components for their needs, e.g., a translator component for their solar water heating system, an alert generator for their car, a storage component to store sensor data on the amazon cloud storage service or an alert manager for sharing alerts on social sites.

3.4.3 Composites network

Currently, a gateway composite is used in the middleware to communicate with a network composed of many nodes. This feature hides the dynamic aspect of the network by ignoring nodes appearance and disappearance. Therefore, the composite can see all node data even if they are encrypted. In fact, it must have privileges to decrypt these data in order to be able to translate and analyse them. In contrast, associating a composite to each node in the network makes each node linked to its own composite which manage its own data. Composites are linked using services or queues. When node N_1 send a message to its composite N_1^v, the message will go through the gateway G_1 and the gateway composite G_1^v, then it reaches N_1^v following the reverse route between N_1 and G_1. Thus, in a security scheme, each physical node can share a key with its composite to secure their communications when the gateway is not trustworthy for example. Let N_1 and N_1^v (G_1 and G_1^v) share a symmetric key K_{N_1} (K_{G_1}, respectively), and $E_K(D)$ means that data D is encrypted using key K. When N_1 and G_1 communicate in one hop, the message flow between N_1 and N_1^v should be as described in Fig. 3.5 where nodes and message data are highlighted in grey and red respectively. This extension can be useful when using an advanced security scheme like the one proposed in Chap. 5.

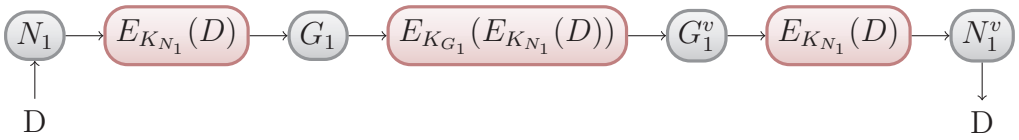


Figure 3.5: Message flow between N_1 and N_1^v.

3.4.4 Social extension

Data sharing between users of the middleware can be very useful in some applications as highlighted in Sec. 2.4. Thus, adding social capabilities in the middleware for the purpose of enabling this feature is highly benefic. It allows organizations to share some sensor data. As an example, the national weather service can share some weather previsions with other users even on existing social web sites or by providing a web mashup that they can integrate in their social portal. Moreover, a collaboration platform between scientists can be easily provided when each group share its sensor data and results with the others. This extension is further discussed in Chap. 6.

3.5 Conclusion

This chapter has presented a new flexible and extensible middleware. This middleware is fully configurable and does not impose a specific data schema, apart from the alert

schema which is not related to data received from sensors. In addition, it sets up sensors, gateways and aggregators in different composites, which ensures basic needs of separation between composites to guarantee data privacy. Moreover, as it is generic, it can be used to replace multiple middleware used in different domains.

This middleware was implemented and used in a real-world project Mobesens, as presented in Chap. 4. A security solution, for securing communications from and to the middleware, is proposed in Chap. 5. Finally, the middleware was extended to enable the building of a social sensor network in Chap. 6.

Chapter 4

A system for storing and visualizing Data from a Sensor Network

Contents

The iSensors middleware described in Chap. 3 is useful, extensible, flexible and can serve in various fields. In our work, we choose the environment field in order to monitor the water quality in the context of the mobesens project. The middleware was used in the back-end system to manage sensor data. The global system was extended by a visualization system for visualizing real-time and history sensor data using any web browser [103, 104].

This chapter presents the Mobesens project and its requirements in Sec. 4.1. The proposed system architecture is described in Sec. 4.2. Then, real-world experiments and performance results are given in Sec. 4.3 and 4.4 respectively. Finally, conclusions are drawn in Sec. 4.5.

4.1 The mobesens project

The Mobesens project [105], which stands for Mobility for Long Term Quality Water Monitoring, aims at developing sensors to measure different physico-chemical parameters

of water so as to appreciate its quality in the environment, and set up both a communication network and a grid infrastructure to allow data to be first transmitted from sensors to the core network and second to be stored, processed and displayed in a convenient and easy-to-use graphical user interface. Note that, even though the project is limited to measure and report water quality parameters, the presented work may be adapted to any other kind of environments, like the soil, the air, etc.

In this project, many wireless sensor networks have been deployed in different areas: Lake Leman, Brest, Thau Lagoon and Ebro River. Each network is composed of CSEM sensor nodes [106] implementing wiseMAC [107] and wiseNET [108] protocols. Each node is connected to one/many physico-chemical sensor via an RS485 interface. Nodes can be mobile or static.

In order to allow the collection and transfer of data for processing, a gateway with broader communication capabilities is placed close to the wireless sensor network. This gateway may either be static or mobile, located on the sea shore or in/on the water. Then, data are transferred to the indexing, storing and eventing system both for persistent storage and to allow more complex processing. Data stored in the system are available through the Internet. Moreover, a visualization interface is provided to help end-users displaying data for monitoring, analysis, etc. This web-based interface was developed to give basic mechanisms to allow various research groups to work together. Furthermore, it is compatible with a variety of devices like tablets, computers, smartphones, etc. as it was developed using web technologies. Fig. 4.1 presents a global overview of the different elements involved in the project and their relations.

The system architecture described in this chapter is integrated with the other elements of this project and is presented in Fig. 4.1 as the GRID element. To summarize, the requirements of the back-end system of the Mobesens project were:

- receiving data from all elements involved in the architecture;
- understanding and transforming received raw data;
- storing raw and transformed data to ensure traceability and persistence;
- tagging and indexing data to facilitate search and reporting;
- real-time alerting to monitor sensor data and network status in a real-time manner;
- enabling the visualization of all these data in a real-time manner from any web-based browser.

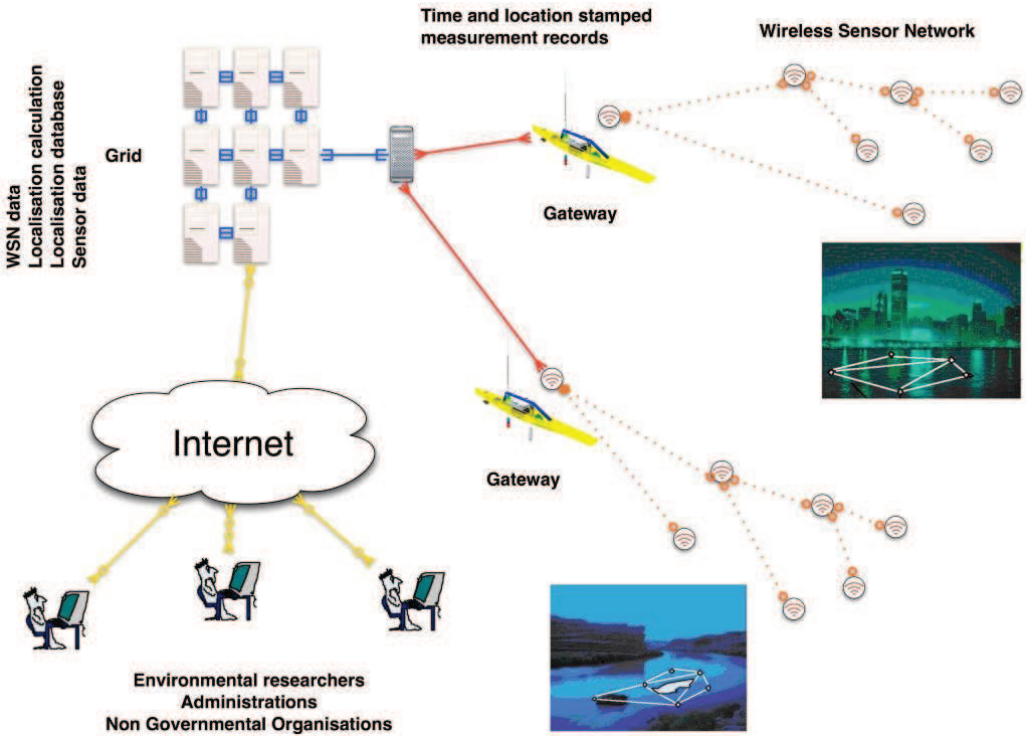


Figure 4.1: Mobesens project overview.

4.2 Back-end and visualization systems description

4.2.1 System Architecture and Implementation

The effective architecture of the system is presented in Fig. 4.2. From this architecture, the back-end and visualization system play an important role in the global system and they take in charge:

- storing data from all elements involved in the architecture, i.e. not only data measured by sensor nodes, but also those used for their management, or any other data which might be used or exchanged by several parties, e.g., the location information about gateways.
- enabling the visualization of all the sensor data in a real-time manner through a web-based application.
- providing a user-friendly web application to generate graphs, browse and validate history data.
- visualizing sensor nodes as anchor points in a map to facilitate their localisation and tracking.

From the sensor nodes point of view, the system shall be seen as a container in which it is possible to store any kind of data regardless the amount and the size. From the

end-users point of view, it shall be seen as a huge collection of organized data on which it is possible to perform high-level requests. In order to satisfy these two points of view and improve the reactivity of the system, data stored by sensor nodes need to be tagged (i.e., they need to be associated with metadata) and indexed before being searched.

The architecture is composed of those elements:

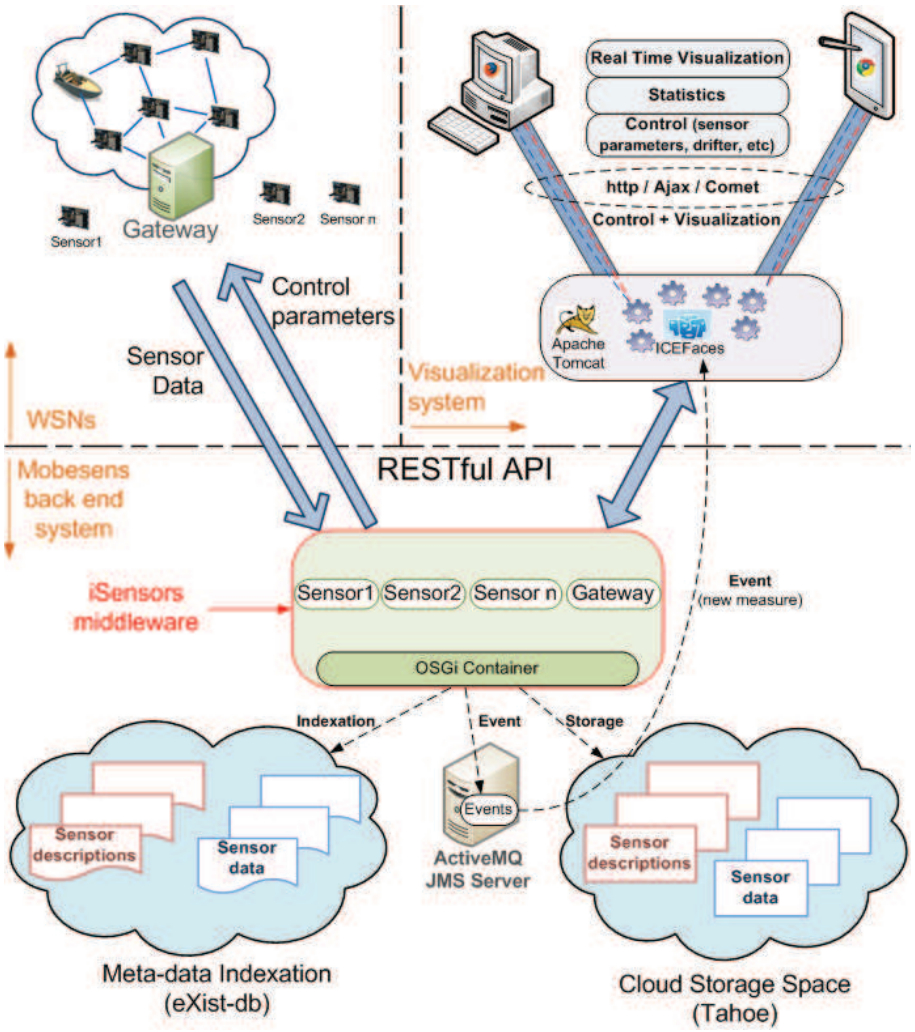


Figure 4.2: Abstract System Architecture and Implementation View.

- The *iSensors middleware* which is described in Chap. 3. Storage and indexation are implemented as different sub-components.
- A *Storage Space* which aims at storing any information. It is used to store both Raw data and XML files. This storage is based on Tahoe [109], a secure cloud storage solution, known for its efficiency and ease of management. It is connected to the Storage sub-component using Secure File Transfet Protocol (SFTP).
- An *Indexation Space* that compiles any metadata provided by sensor nodes, end-users and applications to enable efficient searches. In addition, it indexes the XML

representations of measurements. The tool used to perform the indexation is eXist-db [97] which has been developed for the indexation of XML documents.

- An *Eventing Space* similar to the eventing server as defined in Sec. 3.2.2. This part is one of the key elements inside the back-end system that allows data to be provided to end-users in a real-time manner. This part has been implemented using a standalone Apache ActiveMQ Server, which is an open source message broker implementing the Java Message Service (JMS).

- A *RESTful compliant API* that allows sensors to push their collected data to the system and web-based clients to visualize these data using the same API. Each composite has a communication component, as described in Fig. 3.2a, that provides this API.

- A *Visualization System* responsible for displaying data in a real-time manner and navigating through history data. The home page of the visualization interface contains a map on top and below a data table for each sensor data type in a dedicated tab. Sensor nodes are represented as clickable anchor points in the map. On click, a window, showing information about the sensor node and its recent sensor data, is displayed. The data table displays real-time sensor data ordered from recent to older using the reception date field.

4.2.2 Data processing details

First, raw data are sent from the WSN to the Gateway using IEEE 802.15.4 communication protocol. Then, they are relayed to the back-end system using HTTP and a RESTful API, similar to the easy API described in Table 3.1. As shown in Fig. 4.3, data are processed as follows when received at the Gateway composite in the back-end system:

1. Raw data are stored in the storage space.
2. Raw data are translated to the Metalist XML format [110].
3. The XML version of the data is stored in the storage space.
4. The XML version of the data is indexed in the indexation space.
5. The three identifiers (the stored raw data, and the stored and indexed translated data) are mapped.
6. The mapping file is indexed.
7. The XML version of the data is encapsulated in a message and sent to the ActiveMQ eventing server.

8. The gateway is returned a Unique IDentifier (UID) of the mapping file.

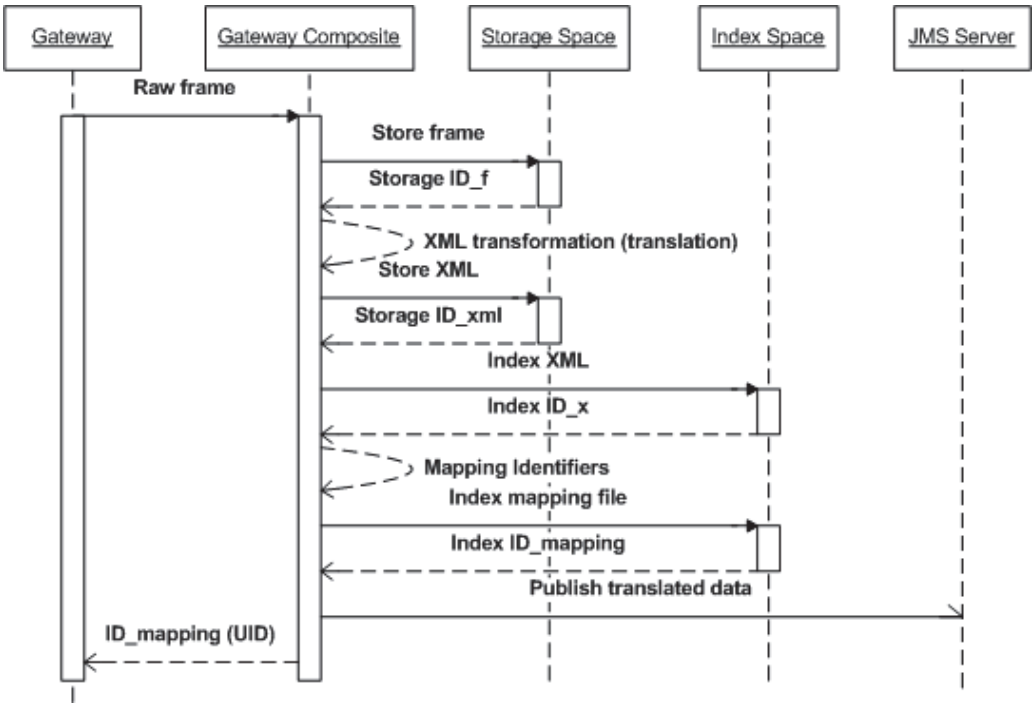


Figure 4.3: Data processing details.

In order to guarantee the interoperability between the gateway, the back-end, the visualization system and third-party applications, a HTTP RESTful interface has been developed. Not only this interface allows the gateway to send data frames to the back-end system, but also enables third-party applications to interact with stored and indexed data.

This interface is generic enough to adapt to new sensors and nodes that may join the network. For example, a node identified by `NodeID` can send a frame to the grid by encapsulating it into a PUT request:

```
http://{ServerAddress}/{Gateway-info}/{SensorType}/NodeID
```

where `SensorType` is a unique identifier associated with each sensor type and a UID is returned as a response to the node. This UID can later be used to retrieve the same data from the system using a GET request:

```
http://{ServerAddress}/{Gateway-info}/UID
```

Data stored in the storage space are persistent. This means that unless they are intentionally removed, data remains in the system regardless the application that stored them is still alive or not. In addition, data are stored in both raw and translated formats with metadata (e.g., source identifier (IP address), reception time, sensor node identifier,

etc.) in order to enable data traceability and persistence. These requirements were fixed by the project to prevent data loss during its translation and to allow scientists to do more extensive analysis in the case of a natural disaster for example.

A dedicated search interface has been made available for the purpose of allowing navigation through data stored and indexed in the system without necessarily knowing their identifiers. This interface receives a query from a user or an application using the Xquery language and returns the list of results that matches the given set of criteria. An XQuery request can be sent to the back-end using the following prototype:

```
http://{ServerAddress}/{Gateway-info}/search?query={Query}
```

4.2.3 Real-time display at the end-user side

Apart from getting a specific piece of data using its UID or searching for a set of data matching a set of criteria, the developed system allows to display the most up-to-date data in a real-time manner. This has been made possible with the introduction of an eventing system based on the combination of an ActiveMQ JMS Server [99] and ICEfaces [111]. This solution allows data to be visualized on the graphical user interface without refresh notification required by the user or client. Moreover, the time needed to update the graphical user interface is very low compared to the frequency of the measurements. This is a key feature since even if data are not updated in a real-time manner in the graphical user interface, the data update is performed quickly enough so that information provided to the end-user is always up-to-date.

Also note that all clients are synchronized, and updates happen concurrently towards all clients. Changes in sensed or stored data automatically trigger updates in all the clients according to their subscriptions and expressed interests.

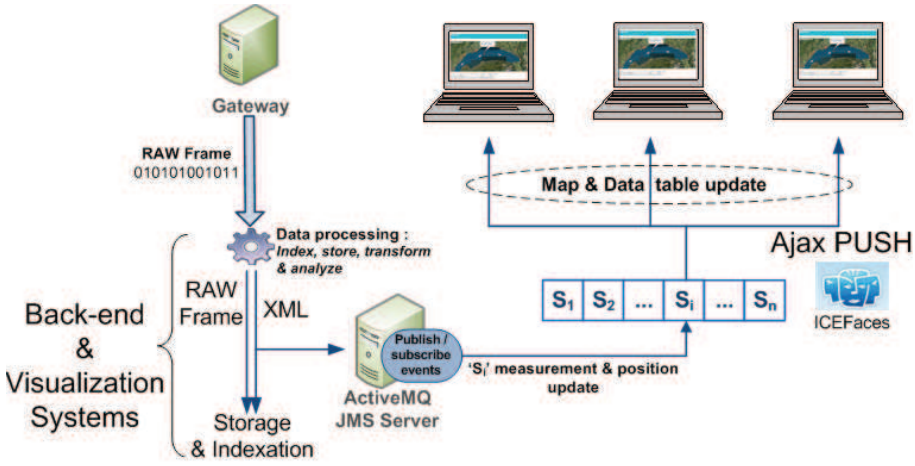


Figure 4.4: Real-time architecture details.

Fig. 4.4 presents the architectural details for the implementation of the real-time

display in the visualization system. The two main elements are the eventing server, e.g., JMS Server, which aims at dealing with any events in the back-end system and ICEFaces that provides an Ajax PUSH link to any registered clients.

At initialization, the visualization system, that implements the Ajax Push components provided by ICEFaces, initiates a persistent communication with the JMS Server and subscribes to topics that match its interest. Then, any time a piece of data is received by the back-end system, the translated (e.g., transformed) version of the data in the XML format is published in the JMS Server as an alert on a specific topic. When events are published on its topics of interest, the visualization system is triggered. Then, it updates or adds new entry in dedicated collection of sensor data S_i values. Next, it informs the ICEFaces core about the update. Once informed, ICEFaces forwards the updates to the browsers (e.g., clients) which are displaying the web interfaces that have subscribed this kind of alerts. For example, a visualization interface displaying temperature measurements receives only temperature updates. Thus, subscribers are notified in a real-time manner and update the visualization interface using the AJAX Push technology. A pseudo code presented in Listing 4.1 is executed in the visualization system server.

```
public void onMessage(Message message) {
 if(message instanceof MeasurementUpdate){
  // Etract information from the message
  parsedMessage = parse(message);
  // Produce a new abstract sensorData object
  // depending on the type and readings of the sensor
  sensorData = SensorDataFactory.getInstance(parsedMessage);
  // Update the list of sensor measurements
  myDataList.update(sensorData.getDataType(), sensorData.getUID(), sensorData);
  // Notify subscribers (Visualization interface)
  SessionRender.render(sensorData.getDataType() + "Topic");
 }
}
```

Listing 4.1: Pseudo-code executed when new alerts arrive at the Visualization system.

The connection between the systems and the clients for the real-time display is performed using the publish/subscribe paradigm. Consequently, the visualization system acts as a subscriber and a publisher of alerts at the same time. It is a subscriber from the point of view of the back-end system, and a publisher from the point of view of the clients. It is the role of this system to establish data synchronization between the new published measurements on the back-end system and the displayed data on the visualization interfaces.

4.3 Real world experiments

The architecture presented above has effectively been developed and is functional. It is possible to see partial results of the visualization interface at this location:

`http://mobesens.free.fr/`

The system is located in Évry, close to Paris, France. The storage space and the indexation space are running on a cluster composed of 32 nodes each including an 8-core Intel Xeon E5540 cluster with 24 GB RAM. The visualization interface has been developed using a Tomcat web application server.

After the entire system has been developed, some real experiments have been conducted. The first one was on Lake Geneva, Switzerland. The second one was in Brest, France. The third one was on the Thau lagoon, France. The fourth one was conducted on the Ebro River, Spain. Finally, the last one was conducted again in Brest with the presence of some members of the European Commission. Fig. 4.5 shows a picture of the kayak and some buoys taken during experiments.

Figure 4.5: Picture of kayak and some buoys taken during tests in Thau lagoon.

The configuration included in the latter experiment:

- 20 buoys, each one equipped with sensor nodes and Ion-Sensitive Field-Effect Transistor (ISFET) [112] sensors to measure temperature and pH.
- a kayak equipped with:
  - a multi-parameter sensor to realize profiles at different depths.
  - a sensor node to forward data to the Gateway.
- a Gateway to collect measured data and send them to the back-end system in Évry through a 3G connection. The gataway is shown in Fig. 4.6, an edited picture taken during experiments in Brest.

Figure 4.6: An edited picture of the Gateway taken during tests in Brest.

During and after the deployment, it has been possible to automatically validate received data (physico-chemical measurements) and visualize them in a real-time manner using any web browser in smartphone, tablet or laptop. In addition, using the real-time interface, available at `http://mobesens.free.fr/`, it was possible to monitor the network, sensor measurements and calibration under a boat in the sea. This has allowed us to detect and quickly solve problems during and after deployment.

4.4 Performance results

The sensor nodes and the gateway communicate using IEEE 802.15.4, wiseMAC and wiseNET protocols while the Gateway and the back-end system communicates using the HTTP protocol. As mentioned above, ISFET sensors are integrated in some sensor nodes. Their measurements are formatted in 37-byte ASCII strings. Sensor nodes send measurements in a single packet to the Gateway over one-or multi-hop routes. The gateway should send an acknowledgement to the sensor node in order to confirm the reception. Then, the Gateway sends the measurement in an HTTP PUT message to the back-end system which confirms by sending an acknowledgement. After storing, transforming and indexing a measurement, the back-end system sends an `UID` as a response to the Gateway which confirms the reception by an acknowledgement. The generated HTTP traffic is composed of two messages and two acknowledgements having a total size of 878 bytes. The traffic has been cut by two by disabling the sending of the `UID`. This way, the size is limited to

547 bytes.

Various experiments have been performed in order to check the performance of this work and especially the ability for the system to provide data in a real-time manner.

In the first test, we stressed our system in order to study how it would behave. The gateway was emulated using a laptop which regularly sent GPS position frames of different nodes to the system. On the same laptop, a browser was opened for the visualization interface to display sending and receiving data times. We used the same laptop to be able to measure the difference between sending and receiving data times using the same clock.

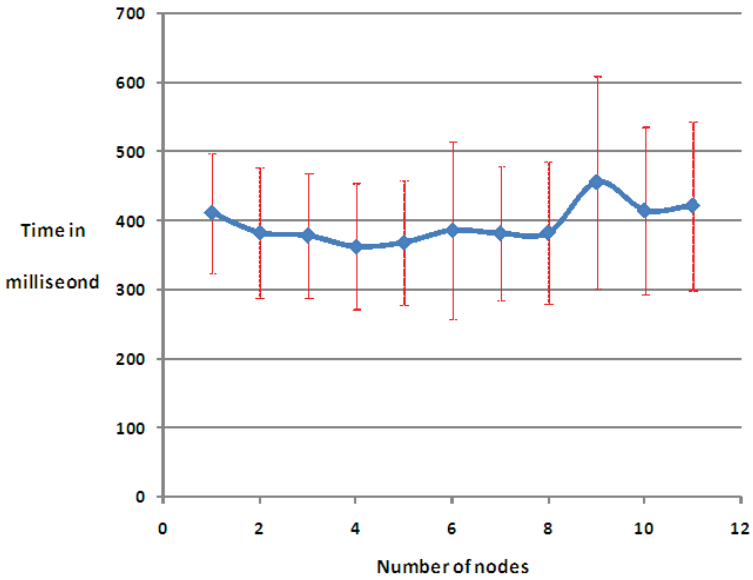
For each experiment, each node sent one hundred messages, e.g., for the experiment with eight nodes, 800 messages was sent to the system. Then, the mean and the standard deviation of the end-to-end delay between sending and receiving frames was computed. Results are presented in Fig. 4.7a. They show that the end-to-end delay is low (less than 500 ms can be considered as real-time for water monitoring as water physico-chemical parameters are changing slowly in the natural environment) and stable.

For the second test, the message size varies. The gateway sends one hundred messages of each size to the system. Fig. 4.7b shows the mean and the standard deviation of the delay between sending the message and receiving the identifier, e.g., the UID in Sec. 4.2.2, for each message size in a logarithmic scale. The default payload length in TinyOS is 29 bytes, so putting a node message in the system takes less than 100 ms which is reasonable compared to the communication delay between the gateway and nodes in the WSN.

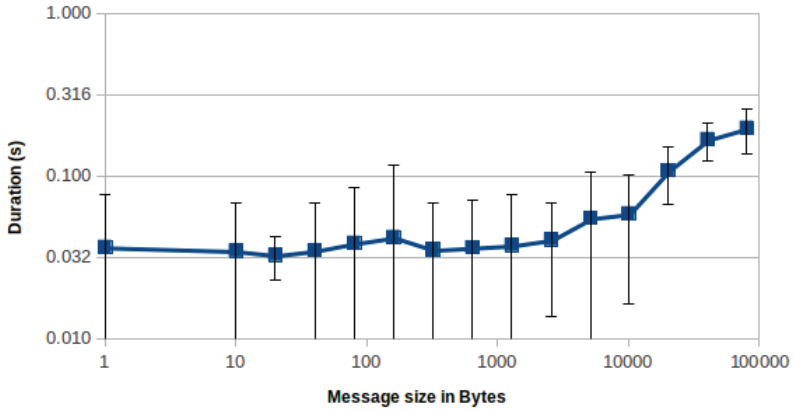
For the third test, the gateway launches several threads at the same time, to send a 40-byte message to the system. Threads emulate nodes that are attempting to send a message at the same time to the system. This experiment is performed one hundred times for each number of threads. Fig. 4.7c displays two curves: the blue one shows the mean and the standard deviation of the delay between sending the message and receiving the identifier in each thread, and the red one shows the mean and the standard deviation of the execution time in the system, e.g., functions represented in Sec 4.2.2. As an example, when 64 nodes start communications with the system at the same time, the blue curve shows that each communication will end after a mean of 450 ms. While, the execution time for storing, indexing and transforming functions in the system, illustrated by the red curve, takes 320 ms. As a result, after a mean of 450 ms all communications have finished and this highlights the system performance.

4.5 Conclusion

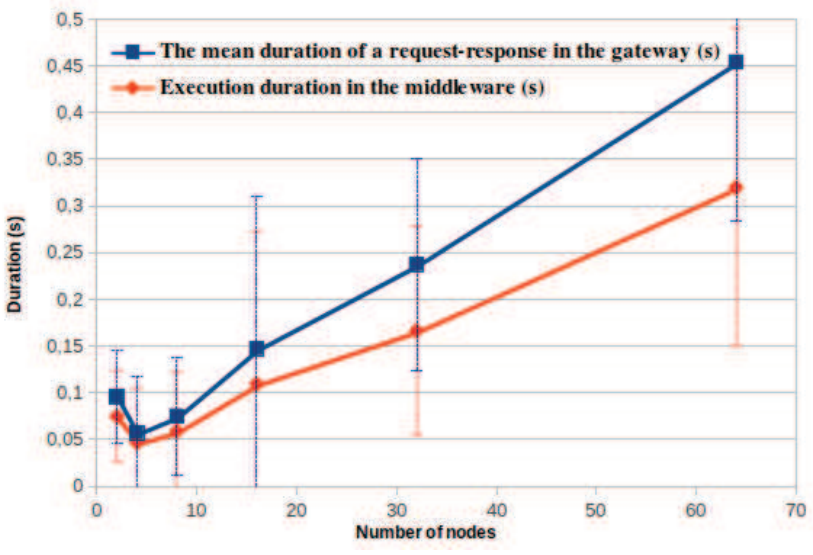
This chapter presented the design and the implementation of a system for water quality monitoring. The system performance was proven using many tests and it has been tested and used many times in real-world experiments. The system provides many features and services for scientists. In addition, third party applications can be easily interfaced with



(a) End-to-end update latency.



(b) The system response delay depending on the message size.



(c) The system response delay depending on the number of nodes.

Figure 4.7: Different performance test graphs.

the back end system due to the use of the RESTful API and the eventing server.

Unfortunately, the security issues of data reception and sharing, which is crucial in such a system, were not treated. Receiving data from the WSN must be secured to prevent an attacker from inserting false data and consequently generating false alerts and altering the integrity of sensor data. Sharing data with other users of the system must also be secured so that each user only sees what he is authorized to see and access. These issues are addressed in the Chap. 5 and Chap. 6 respectively.

Chapter 5

Hybrid Security for WBANs

Contents

The increase in living standards, the growth of aging population and health care costs on one side and the electronics and wireless communication advances on the other side triggered the development of new kinds of networks, named Wireless Body Area Networks (WBANs), to remotely monitor patient. WBANs have some similarities with wireless sensor networks (WSNs) such as low computation capacity, small amount of memory and limited energy resources [113].

A WBAN consists of small intelligent sensor and/or actuator nodes attached or implemented in the user body which are communicating wirelessly with a personal Local Processing Unit (LPU), eg., a PDA or a smartphone. In the remaining of the chapter,

we note sensor and/or actuator nodes as SN and LPU as a Mobile Node (MN). MN acts as a sink for data of SN and forwards data to a Storage Site (SS) where they are stored and analyzed [114, 115]. These SNs provide continuous health monitoring and feedback to the user or medical personnel. SS can be seen as the gateway composite in the iSensors middleware and for more simplicity it is considered in the rest of the chapter as the middleware.

Sensor and/or actuator nodes have extremely scarce resources in term of memory, energy and storage. They communicate wirelessly with the MN using either ZigBee or Bluetooth [113]. Although MN has more resources, they remains limited as it uses a battery and wireless networks. As presented in [24, 25], sensor nodes are used to monitor many vital signs like temperature, heart rate, breathing rate, blood pressure, electrocardiogram, etc. Actuators take some specific actions according to the data they receive from sensors or through interactions with the patient like pumping the dose of insulin to diabetics based on glucose level measurements.

The iSensors middleware described in Chap. 3 can be very useful for a user or organiation to manage their WBAN data. However, these data can encapsulate sensitive and private information and thus strictly deprived. In addition, the middleware must avoid the anonymous push of data in order to prevent attackers from inserting false sensor data. This chapter presents a novel security scheme to fulfil these requirements [116].

The network architecture, as presented in Fig. 5.1, is composed of many clusters. Each one is headed by an MN and other members are SNs which can communicate in a single-hop with their associated MN. The cluster head can communicate with SS using any Internet connection: GPRS, 3G, wifi, etc. In order to differentiate those two communication types, they were classified as *intra-body communication* (IbC) and *extra-body communication* (EbC) [25]. IbC and EbC are interconnected through the MN. IbC is composed of SNs with scarce resources while EbC is composed of MNs and an SS where resources are more abundant: higher communication bit rate, processing and storage capabilities. Besides their hardware difference, these two communication types encounter different security threats. Indeed, SNs and their associated MN belong to a user, which makes an attacker unable to easily tamper these nodes. In addition, the small radius in which IbC operates ($\sim$2 m) facilitates the preservation of security and privacy [117]. In contrast, EbC is transported over the Internet which makes security threats more important.

It is important to realize that IbC and EbC have different properties and need different security mechanisms. This chapter proposes a hybrid authentication and key establishment scheme that enables mutual authentication and key establishment between all entities composing a WBAN, eg. SN, MN and SS. This is based on two protocols. The intent of the first one is the mutual authentication between SS and MN, by providing an asymmetric pair of keys for MN, and by establishing a pairwise key between them. The

second one aims at authenticating them, and at establishing a group key and pairwise keys between SN and the two others. Note that the resulted symmetric keys can serve as master keys for key derivation functions (KDF) [118] or the μ-Tesla protocol [119] for advanced key management scheme. The hybrid property is used to highlight the fact that asymmetric keys are generated and used to sign messages in the EbC while only symmetric keys are generated and used in the IbC.

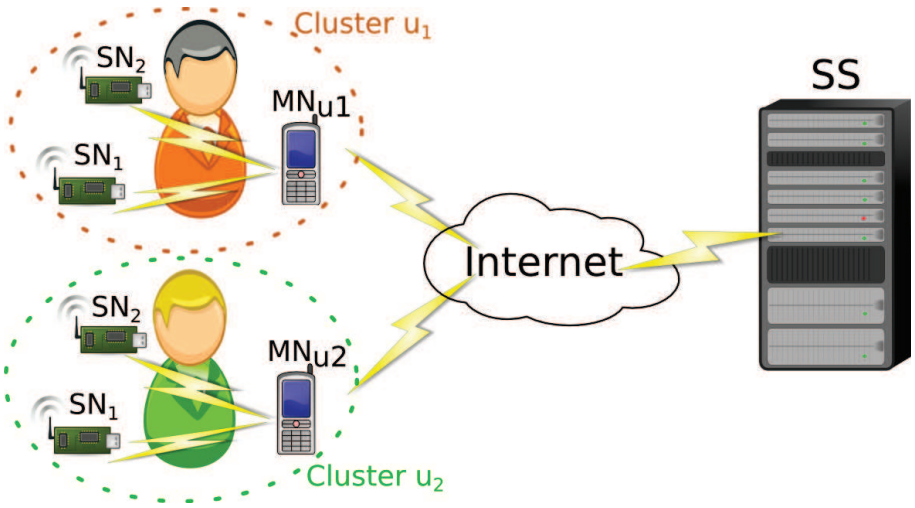


Figure 5.1: WBAN architecture.

Note that the traffic between the middleware and the physical nodes takes two forms. In the first form, the physical gateway node interconnects a set of physical sensor nodes to the gateway composite. The security can be applied either only between the gateway node and composites or between sensor nodes, the gateway node and gateway composites. In the second form, the communication is performed between sensor nodes and sensor composites directly which makes it similar to the case where communication occurs between the gateway node and composites. Thus, similar security mechanisms can be applied in this form.

The rest of the chapter is organized as follows: security primitives and the WBAN architecture are presented in Sec. 5.1. Sec. 5.2 describes the two protocols for authentication and key establishment. Security and performance analysis of the presented scheme are the subject of Sec. 5.3. Sec. 5.4 presents some related works. Finally, conclusions are drawn in Sec. 5.5.

5.1 Background

This section presents some security primitives that are used to prove the efficiency of the proposed scheme in Sec. 5.2.

5.1.1 Identity-based Signature

Identity-based cryptography (IBC) is a type of asymmetric key-based cryptography in which a publicly known string representing an individual or an object is used as a public key. The private key is related to the identity too. There is no need for a certificate, a Public Key Infrastructure (PKI) or a Certification Authority (CA). This makes IBC more suitable for nodes having limited resources (energy and memory). A complete comparison between IBC and PKI is presented in [120]. The concept of IBC was first introduced by Adi Shamir in 1984 [121].

Based on this work, many researches have proposed practical signing and/or encrypting schemes. Hess proposed an ID-based signature scheme (IBS) using ECC and based on pairings [122]. The security relies on the hardness of the W-DH problem. The scheme consists in four algorithms (*Setup*, *KeyGen*, *Sign* and *Verify*) and three parties (the trusted Private Key Generator (PKG), the signer and the verifier).

Let $(G_1, +)$ and $(G_2, .)$ be two cyclic groups of prime order q, $P \in G_1$ be a generator of G_1 and $\hat{e} : G_1 \times G_1 \rightarrow G_2$ be a pairing which satisfies the following conditions:

1. ***Bilinearity:*** $\hat{e}(aX, bY) = \hat{e}(X, Y)^{ab}$ for all $X, Y \in G_1$ and $a, b \in \mathbb{Z}_q^*$.
2. ***Non degenerate:*** There exists $x \in G_1$ and $y \in G_1$ such that $\hat{e}(x, y) \neq 1$.

Assume that for any given random $b \in G_1$ and $c \in G_2$, it should be infeasible to compute $x \in G_1$ such that $\hat{e}(x, b) = c$. Let the two hash functions h and H_1 be defined as $h : \{0,1\}^* \times G_2 \rightarrow \mathbb{Z}_q^*$ and $H_1 : \{0,1\}^* \rightarrow G_1^*$. In this context, the public parameters are $params = (G_1, G_2, \hat{e}, q, P, h, H_1)$ and algorithms can be specified as follows:

- ***Setup:*** The PKG picks a random integer $s \in \mathbb{Z}_q^*$ as the local secret value, computes $P_{pub} = s.P$ and publishes P_{pub}.
- ***KeyGen:*** The key generation starts when the PKG receives the ID of the signer (let the signer's identity be string ID). The private key of the signer is then given by $Priv_{ID} = s.H_1(ID)$. It is computed by the PKG and given to the signer. The public key of the signer is $H_1(ID)$.
- ***Sign:*** To sign message m, the signer chooses an arbitrary point $P_1 \in G_1^*$ and a random integer $k \in \mathbb{Z}_q^*$ and executes the following steps:
  1. $r = \hat{e}(P_1, P)^k$
  2. $v = h(M, r)$
  3. $U = v.Priv_{ID} + k.P_1$

  Then, the signature is the pair $(U, v) \in G_1 \times \mathbb{Z}_q^*$

- ***Verify:*** On the reception of message m and signature (U, v) the verifier computes:

  1. $r = \hat{e}(U, P).\hat{e}(Pub_{ID}, -P_{pub})^v$
  2. the signature is accepted if and only if $v = h(m, r)$

5.1.2 Diffie Hellman Assumptions

Developed in 1976, the Diffie Hellman method for exchanging cryptographic keys has been extensively used to allow two entities to generate a shared secret to securely exchange data over an non-secure communication link. Its efficiency is based on the assumption that some mathematical problems are very difficult to solve. To prove the effectiveness of the security protocol, we present some Diffie Hellman problems.

5.1.2.1 The Weak Diffie-Hellman (W-DH) problem

Given $P, Q, sP \in G_1$ for some $s \in \mathbb{Z}_q^*$, it is hard to compute sQ.

5.1.2.2 The Computational Diffie-Hellman (C-DH) problem

Given $P, aP, bP \in G_1$ for some $a, b \in \mathbb{Z}_q^*$, it is hard to compute abP.

5.1.2.3 The Bilinear Diffie-Hellman (B-DH) problem

Given $P, aP, bP, cP \in G_1$ for some $a, b, c \in \mathbb{Z}_q^*$, it is hard to compute $\hat{e}(P, P)^{abc} \in G_2$.

5.1.3 Joux Key Agreement

The Joux protocol provides a three party key agreement protocol [123] based on the Bilinear Diffie-Hellman (B-DH) problem. Assume three nodes A, B and D generate K_A, K_B and K_D respectively, which are parts of a common key. Each node broadcasts its part multiplied by P to the others. Then, common key K_c is computed as follows:

- A computes $K_1 = \hat{e}(K_B.P, K_D.P)^{K_A}$
- B computes $K_2 = \hat{e}(K_A.P, K_D.P)^{K_B}$
- D computes $K_3 = \hat{e}(K_A.P, K_B.P)^{K_D}$

In fact, the common agreed key is given by

$$K_c = K_1 = K_2 = K_3 = \hat{e}(P, P)^{K_A.K_B.K_D}$$

5.2 Authentication and key establishment scheme

The proposed protocol in this section uses three types of keys. First, asymmetric pair of keys are generated using Hess scheme [122] presented in Sec. 5.1.1. Second, pair wise keys are generated based on the Computational Diffie-Hellman (C-DH) problem. Third, a three-party key is based on Joux key agreement described in Sec. 5.1.3.

The protocol can be divided into two parts. The first one focuses on the authentication and key establishment in the EbC. It lets MN get public and private keys using Identity based cryptography (IBC) after an authentication process with SS which is supposed to be the PKG. PKG can be a specific composite in the iSensors middleware, thus it can be reduced on the SS as this latter is considered as the middleware. A pairwise key is also established in this part based on Elliptic Curve Diffie–Hellman (ECDH). In the second part, SN, which is joining the network, authenticates itself to MN and SS, and establishes pairwise keys with each of them and a three party key.

5.2.1 Prerequisite

MN, SN and SS are synchronized and the drift between clocks should not exceed a given threshold (Δ in EbC and Δ' in IbC), specified in the system and stored in all nodes. This condition is not crucial but gives a better efficiency for the scheme against replay attacks. Assume that hash function H is sufficiently secure.

5.2.2 Setup

Each node SN or MN is identified by a node ID, a user ID and password *pwd*, all stored in SS and setup in nodes at initialization. The ID of MN must be unique in the system. The ID of an SN must be unique in the set of user's SNs. The *pwd* is private for each node and must not be stored in the MN and entered by the user at the beginning of its authentication process. All identifiers and passwords are securely stored in SS.

SS plays the role of the PKG, so it generates public parameters *params* which are then stored in MNs and SNs.

5.2.3 MN-SS authentication and key establishment scheme

This scheme ensures the authentication and key establishment between MN and SS. At its end, if MN and SS are successfully authenticated, MN must obtain an asymmetric pair of keys and share a pairwise key with SS. It is presented in Fig. 5.2 and can be divided into five steps as described below:

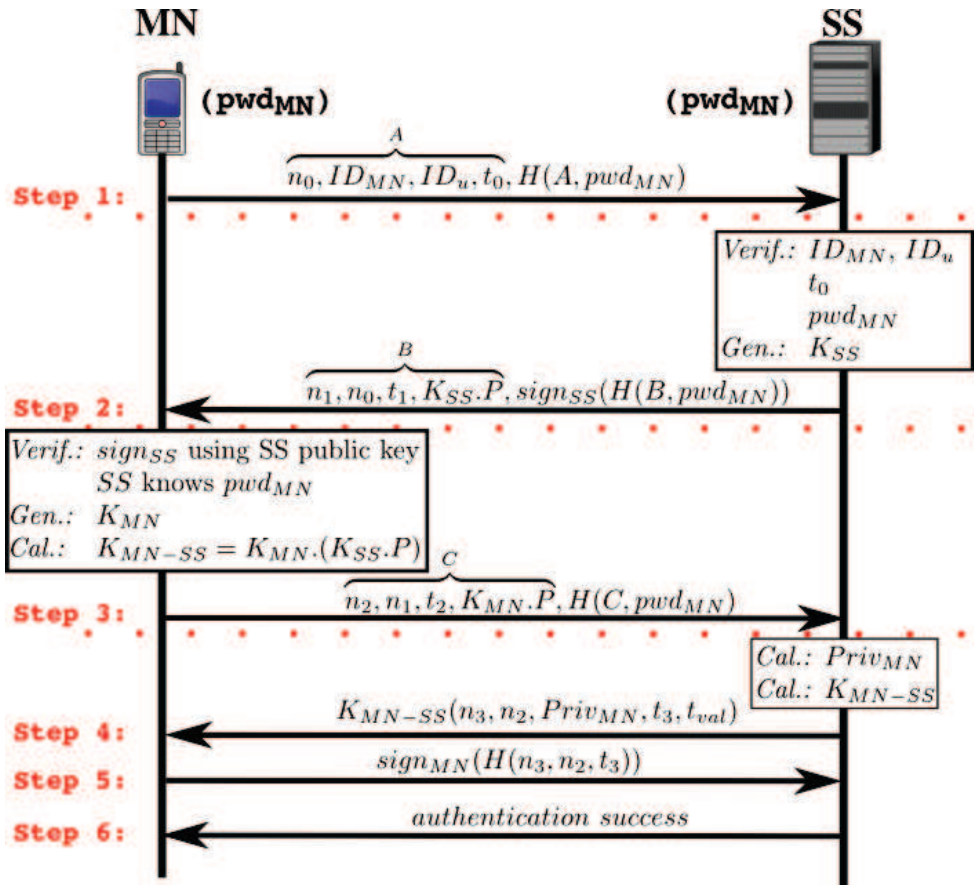


Figure 5.2: MN-SS authentication and key establishment scheme.

5.2.3.1 Step 1

Objectives : MN wants to authenticate itself to SS, so it must prove its identity using a pre-shared password.

Details : MN starts the process by sending a start authentication message containing a nonce value n_0, its identity ID_{MN}, its user identity ID_u and time stamp t_0; all those values and their associated password pwd_{MN} are hashed and the result is added to the message footer. Then, the message is sent. A nonce value and a time stamp are provided in the message to enforce its resiliency against replay attacks. pwd_{MN} is included along with the hashed values to let SS authenticate MN and prevent Deny of Service (DoS) attacks.

5.2.3.2 Step 2

Objectives : The goal for SS is (i) to verify MN identity, (ii) to prove its identity, (iii) to generate and to share its part ($K_{SS}.P$) of the pairwise key K_{MN-SS} with MN.

Details : On receiving this message, SS checks identities ID_{MN} and ID_u to ensure that MN is allowed to integrate the network. Then, it checks that the difference between the actual time and t_0 is smaller than Δ. Finally, it checks that MN has the corresponding pwd_{MN} by verifying the result of the hash function.

If the verification succeeds, SS generates its part K_{SS} of the Diffie–Hellman (DH) key. Next, it generates a message containing the received nonce value n_0, a new one n_1, time stamp t_1, and its part of the DH key multiplied by point P ($K_{SS}.P$). The hash of pwd_{MN} and all the fields using the private key are signed and concatenated to the signature of the message. The three first fields are used to establish the message sequencing and provide resiliency against replay attacks. pwd_{MN} is included in the hash computation to let SS argue that it knows it, so it is not an attacker. Note that signing a hash is more secure than signing clear fields to prevent attacker from having the pwd_{MN}.

5.2.3.3 Step 3

Objectives : The goal for MN is (i) to authenticate SS, (ii) to generate its part ($K_{SS}.P$) of the pairwise key K_{MN-SS}, (iii) to compute the key, (iv) and to share its part with SS.

Details : On receiving the message from SS, MN checks the freshness of the message, the authenticity of SS and that this latter knows pwd_{MN}. As a result, MN checks that n_0 is present in the message and that the difference between the actual time and t_1 is smaller than Δ. Then, MN checks SS authenticity and knowledge of pwd_{MN} by verifying the signature using this latter public key.

Success in verification means that the message is an authentic and fresh response from SS, otherwise the authentication fails. When authentication succeeded, MN generates its part K_{MN} of the DH key K_{MN-SS} and computes it using equation $K_{MN-SS} = K_{MN}.(K_{SS}.P)$. Finally, it generates a message containing n_1, a new nonce n_2, time stamp t_2, $K_{MN}.P$ and pwd_{MN}. Then the message is sent to SS after including the hash result of all the fields and its pwd_{MN}.

5.2.3.4 Step 4

Objectives : The goal for SS is (i) to get the part of MN ($K_{SS}.P$), (ii) to generate the pairwise key K_{MN-SS}, (iii) to generate the private key of MN, (iv) and to send it to MN encrypted using K_{MN-SS} in order to verify that this key was successfully computed.

Details : On receiving the message from MN, SS checks the hash result, time stamp t_2 and the presence of n_1. Then, it gets $K_{MN}.P$ from the message and computes the DH key using equation $K_{MN-SS} = K_{SS}.(K_{MN}.P)$, and generates the private key of MN ($Priv_{MN}$). Finally, SS creates a message containing the received nonce n_2, a new one n_3,

$Priv_{MN}$, time stamp t_3 and the validity time t_{val} of the private key. All the fields are encrypted using DH key K_{MN-SS} and the messages is sent to MN.

5.2.3.5 Step 5

Objectives : The goal for MN is (i) to get its private key, (ii) and to prove that the correct pairwise key was successfully computed.

Details : MN decrypts the message, checks t_3 and the presence of n_2. MN gets its private key and generates a message containing the hash of n_2, n_3 and t_3, signs it using its private key and sends it back to SS.

5.2.3.6 Step 6

Objectives : SS checks that MN received its private key successfully.

Details : SS checks the message fields and the signature to make sure that MN received its private key successfully. If the verification succeeds, SS sends an *authentication success* message to MN to end the authentication and key establishment protocol; otherwise the message is ignored.

5.2.4 SN-MN-SS authentication and key establishment scheme

After establishing a secure link between MN and SS thanks to the defined protocol in Sec. 5.2.3, connected SNs to MN must authenticate themselves to both MN and SS before being able to communicate with them. The protocol, presented in this section, aims at providing a secure scheme to let SN securely establish a three-party key and two pairwise keys, each one being shared with MN and SS respectively. Running the protocol presented in Fig. 5.3 needs four steps which are described below.

5.2.4.1 Step 1

Objectives : SN wants to authenticate itself to MN and SS, so it must prove its identity using a pre-shared password with SS. MN generates its part ($K_{MN}.P$) of both the pairwise key K_{SN-MN} and the three-party key.

Details : SN starts the protocol by sending a message to MN including nonce value n_0, its identity ID_{SN}, its user identity ID_u and time stamp t_0; it hashes all these fields and its pwd_{SN} shared with SS, then it includes the resulted hash to the message and sends it to MN.

MN checks if the time stamp t_0 is fresh. If it is the case, MN stores the three first fields and generates its part K_{MN} of DH key K_{SN-MN} to be shared with SN. Next, it

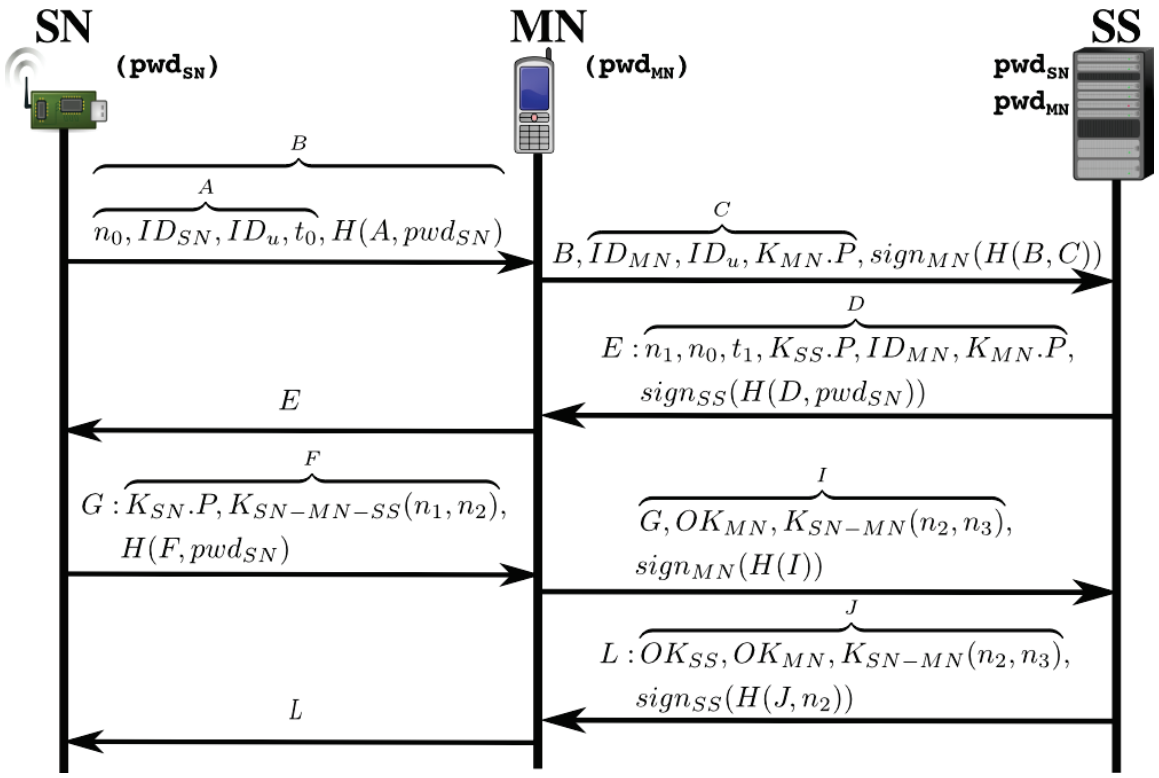


Figure 5.3: SN-MN-SS authentication and key establishment scheme.

adds its identity ID_{MN}, its user identity ID_u and $K_{MN}.P$ to the received message from SN, it hashes all the fields and signs the hash using its private key $Priv_{MN}$. Finally, it adds the signature to the message footer and sends it to SS.

5.2.4.2 Step 2

Objectives : The goal for SS is (i) to verify SN and MN identities, (ii) to generate and share its part ($K_{SS}.P$) of both the pairwise key K_{SN-SS} and the three-party key with MN and SS, (iii) and to confirm the authenticity of MN and its part ($K_{MN}.P$) to SN. In addition, MN verifies the authenticity of SN.

Details : On receiving the message from MN, SS checks all the identities in the message and the fact that the time stamp fulfill inequation $|t - t_0| < \Delta + \Delta'$. Then, it checks the signature of MN using Pub_{MN} and the hash included in message B to verify that SN has the correct pwd_{MN} and the message was not modified during routing.

If all verifications succeed, SS generates its part K_{SS} of DH key K_{SN-SS} before generating a response to SN, which contains received nonce n_0, a new one n_1, time stamp t_1, $K_{SS}.P$, MN identity to inform SN that it is an authorized node, and $K_{MN}.P$. It signs the hash of all those fields and pwd_{SN} using its private key $Priv_{SS}$ and adds the signature to the message footer. Finally, it sends the message to SN.

5.2.4.3 Step 3

Objectives : The goal for SN is (i) to verify the authenticity of SS and MN, (ii) to generate and share its part ($K_{SN}.P$) of the two pairwise key (K_{SN-SS} and K_{SN-MN}) and the three-party key with MN and SS, (iii) to compute the three keys, (iv) and to prove that the correct keys were successfully computed. Moreover, MN computes the two keys and proves their rightness.

Details : First SN checks the availability of n_0 and ID_{MN}, the validity of t_1, and the signature. If this verification succeeds, SN generates its part (K_{SN}) of the two DH keys and the three party key and then computes these three keys. After SN have picked nonce value n_2, it encrypts n_1 and n_2 using common key $K_{SN-MN-SS}$ to get $e_1 = K_{SN-MN-SS}(n_1, n_2)$. Finally, it builds message G using its part of DH keys multiplied by point P, e.g. $K_{SN}.P$, and e_1. It adds the result of hashing all the fields and pwd_{SN} to the message and sends it to MN.

MN gets the part of SN (K_{SN}) in the DH pairwise key (K_{SN-MN}) and the common key ($K_{SN-MN-SS}$) and generates the two keys. Then it decrypts the encrypted field with this latter key and checks that n_1 is available. It gets n_2, picks a new nonce n_3 and encrypts them with key $K_{SN-MN-SS}$. It adds the result with the flag OK_{MN} and signs the received message and the new fields using its private key $Priv_{MN}$. Finally, it sends the resulted message to SS.

5.2.4.4 Step 4

Objectives : The goal for SS is (i) to compute the three keys, (ii) and to prove to SN and MN the success of the authentication and key establishment process. Let MN and SN validate the authentication process.

Details : SS checks both the signature of MN using the latter public key and the availability of OK_{MN} in the message. If the verification succeeds, it gets the part of SN $K_{SN}.P$ and computes twice K_{SN-SS} and $K_{SN-MN-SS}$. It decrypts the encrypted field in the message and verifies the existence of n_1. If the verification succeeded, it gets n_2 and sends a message to SN containing OK_{SS}, OK_{MN}, the encrypted values sent by MN in message I. All these fields and n_2 are hashed then signed before being sent.

MN just checks that SS sent OK_{SS}, OK_{MN}, the encrypted fields, the SS signature and forwards the message to SN.

On receiving the message, SN checks the signature of the message, that both SS and MN had sent an OK value. Finally, it checks the encrypted values of n_2 and n_3. If verifications succeeded, the authentication and key establishment are valid too.

5.3 Analysis of our scheme

5.3.1 Security analysis

The protocol described in Sec. 5.2.4 shows that MN cannot check the hash of message B and the signature in message E. Therefore, they can be forged by an attacker to realize a DoS attack on this node. In order to minimize the threat of forging message B, SS must send to MN the list of identifiers of all the user's authorized SNs, after the success of the protocol presented in Sec. 5.2.3. This list should be dynamic and updated when modifications occur. Moreover, SS must sign message E using its private key or encrypt it using pairwise key K_{MN-SS} for the purpose of allowing MN to check SS authenticity.

The rest of the section is devoted to describe the resiliency of the aforementioned protocols against some attacks.

Denial of Service attack (DoS)

Besides the aforementioned enhancements to avoid an attacker from sending a big amount of authentication requests to MN, received requests should be limited to some threshold on SS to avoid a distributed DoS (DDoS) attack. In fact, an attacker may control many MNs and launches a DDoS attack against the SS by sending different authentication request messages from both MNs and SNs. The aim of the attack is to flood the SS with a big amount of authentication requests in the intention of making SS unable to process legitimate requests. In order to avoid such an attack, SS should limit both the number of authentication requests of new MNs to a given threshold T_0 during a time slot Δ_{T_0} and the number of requests of new SNs from each MN to T_1 ($\leq$ the maximum number of SNs per user) during time slot Δ_{T_1}. In addition, the use of the MN signature helps the SS to authenticate the message origin and to make sure the freshness is correct thanks to the timestamp.

Replay attack

In such attack, a malicious node repeats or delays the transmission of valid messages. In order to prevent it, exchanged messages in both protocols include nonce integers and time stamps to establish message sequencing and prove message freshness. Additionally, the user identifier is included in authentication request messages. Then, in all upcoming messages, every nonce integer is associated with the one that has been sent in the previous message. For example in message E (see Fig. 5.3), n_0 is associated with n_1 and in message G, n_1 is associated with n_2. So, an attacker is able either to impersonate the MN or the SN. However, it is not able to replay old messages. In addition, if it catches a message in a cluster and tries to replay it in another one, it fails because ID_u is different.

Man in the middle attack

To perform this attack, the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other, when in fact the entire communication is controlled by the attacker who can forge and inject new messages. Attack succeed when the attacker can impersonate each endpoint to the satisfaction of the other. In order to prevent it, both protocols use password pwd and/or signatures which makes message forgery impossible. In the protocol defined in Sec. 5.2.3, when an attacker attempts a man in the middle attack, it needs to get pwd_{MN} in order to masquerade the SS from one side and the MN from the other side. As pwd_{MN} is kept secret and sent hashed, the attacker is not able to get it unless a brute force attack is performed. The same conditions occur in the protocol defined in Sec. 5.2.4.

5.3.2 Performance Analysis

In order to analyze the performance of this scheme, the main operations and the number of times they are used in each step are countered. It is important to consider pairing function $\hat{e}$, as its complexity is $O(log_2 p)$ [124]. A multiplication also is a time-consuming operation as a 160-bit ECC point multiplication takes about 0.81 s on an 8-bit microcontroller [125]. Exponentiation and hashing functions are also considered in the analysis while symmetric encryption is ignored for the reason that it does not take a lot of time and usually is optimized using a hardware accelerator. Considering the optimization proposed in [126] and the fact that data are hashed before they are signed, the signature operation requires one exponentiation, one multiplication and two hash-function evaluations. However, the verification requires one exponentiation, two hash-function evaluations and one paring $\hat{e}$. The cost of both protocols defined in Sec. 5.2.3 and 5.2.4 are given in Tables 5.1 and 5.2 respectively. We deduce from these tables that the computation load is more important in the SS side in both protocols. Table 5.2 shows that SN computation load is low although it performs the authentication of two nodes MN and SS because SS checks the authenticity of MN messages and includes relevant information in its messages destinated to SN. Thus, SN only checks signatures of SS.

5.4 Related work

Authors in [115] propose an IBC scheme to secure access to patient data. In this scheme, each SN periodically generates, e.g. each day or each hour, a new public key using a string $str = \{date|time|ER\}$, where ER is the monitored vital sign. SN encrypts each

[1] $\hat{e}$ pairing evaluation
[2] Exponentiation
[3] Multiplication
[4] Hash

	MN				SS			
	$\hat{e}^1$	E^2	M^3	H^4	$\hat{e}$	E	M	H
step 1	0	0	0	1	0	0	0	0
step 2	0	0	0	0	0	1	2	3
step 3	1	1	1	3	0	0	0	0
step 4	0	0	0	0	0	0	2	2
step 5	0	1	1	2	0	0	0	0
step 6	0	0	0	0	1	1	0	2
SUM	1	2	2	6	1	2	4	7

Table 5.1: Computation cost in MN-SS authentication and key establishment scheme.

	SN				MN				SS			
	$\hat{e}$	E	M	H	$\hat{e}$	E	M	H	$\hat{e}$	E	M	H
step 1	0	0	0	1	0	1	2	2	0	0	0	0
step 2	0	0	0	0	0	0	0	0	1	2	2	4
step 3	1	2	3	3	0	2	3	2	0	0	0	0
step 4	1	1	0	2	1	1	0	2	1	3	3	4
SUM	2	3	3	6	1	4	5	6	2	5	5	8

Table 5.2: Computation cost in SN-MN-SS authentication and key establishment scheme.

measurement using this public key to get tuple (c_1, c_2) which is later sent to MN where tuples are aggregated to form set $\{(c_1^1, c_2^1), \ldots, (c_1^k, c_2^k)\}$. MN sends the sets to SS. A doctor, willing to obtain data collected under some *str*, first gets permission from the CA. After the CA agreement, it derives the corresponding private key needed to decrypt data and sends it back to the doctor. Finally, the doctor retrieves the data from SS. This solution is interesting since it protects SN data using IBC and does not generate any communication overhead for the key management. However, it has some drawbacks:

- the need for a CA despite the use of IBC.
- data arrive encrypted to SS which makes realtime data analysis impossible.
- data querying operation is very heavy.
- the attacker can simply inject forged messages in the network as data are encrypted using the public key. Thus, any attacker having IBC public parameters and the *str* syntax can generate the public key, generate a legitimate message and inject it in the network as data are sent encrypted and no access control or authentication mechanism is performed.

Authors in [127] and [128] have focused only on security between SN and MN. M. Barua et al. [127] classify all data packets into two major categories, high and low priority classes. Then, IBC schemes are used to authenticate, encrypt and decrypt packets. J. Liu et al. [128] use IBC, when there is not a pre-shared master key, to establish a symmetric session key between SN and MN.

The use of IBC in mesh networks has been proposed as a practical scheme to authenticate stations to Access Points in the first hand and to enable authentication between stations on the other hand [129]. Authors also propose some extensions to update asymmetric keys and adapt the protocol to different network architectures. In [130], an identity based key agreement and encryption scheme were proposed but evaluation results show that encryption and decryption with 160-bit key requires 6.8s and 5.2s respectively which makes IBC not suitable to be used in frequent operations in a WBAN where messages should be delivered in reasonable time to SS [25].

5.5 Conclusion

This chapter presented a hybrid authentication and key-establishment scheme which combines symmetric cryptography and identity-based cryptography. Nodes having scarce resources use symmetric keys, while those having more resources use asymmetric keys. The identity-based signature concept has been chosen as it offers a simpler private key management and generation system than a traditional Public Key Infrastructure. The security and performance analysis shows that our scheme is resilient against known attacks and that computation load is reduced on SNs due to MNs authentication done by SS. In fact, this chapter presented a security scheme to secure the communication between physical nodes and the middleware. Securing the sharing and distribution of sensor data is described in the next chapter.

Chapter 6

Social sensor networks

Contents

Social networking sites have penetrated the lives of Internet users. They allow collaboration, sharing, making social relationships, etc. through web-based interfaces. Social mashup engines enable building personalized web portals and sharing applications.

Chap. 3 presented the iSensors middleware which gives users or organizations the ability to have a private space to manage their heterogeneous sensor data. Therefore, users should have, maintain and administrate an online server where the middleware is installed to real-time collect data from the connected nodes. This is difficult for an ordinary user. Thus, providing a system that allows each user to have its private middleware in an online server while offering the administration and maintenance of the server is a today concern. Moreover, in order to interpret and analyse sensor data, the system should be shared by different collaborators, for example by scientists insterested on monitoring an environmental phenomena or by patients and doctors. When considering the issues of

sensor data sharing on user privacy, the aforementioned requirements lead to significant challenges.

This chapter describes in Sec. 6.1 a cloud infrastructure for sensor networks where each user can have its own server, equiped with the middleware, in the cloud. Sec. 6.2 describes a social networking extension for the iSensors middleware. The security challenges of data sharing in social sensor networks are handled in Sec. 6.3.1. The conclusion is provided in Sec. 6.4.

6.1 Cloud infrastructure for sensor networks

6.1.1 Motivations

Each user/organization has to use, manage and administrate its own instance of the iSensors middleware, as defined in Chap. 3. As a result, the middleware can be deployed as a SaaS (Software as a Service) in the cloud for the purpose of making it able to manage data coming from many users at the same time while keeping its advantages. Deploying an instance of the middleware for each user in the cloud is a good solution for many reasons:

- connect different devices and networks.
- storage of data and index it on the cloud.
- easy to add new users/organizations.
- use services and do not care about infrastructure.
- the high performance and availability.

6.1.2 Architecture

In order to make the iSensors middleware available on the cloud for many users, the architecture described in Fig. 6.1 is used. The cloud infrastructure is managed using the OpenNebula open-source software which offers rich solution for complete management of virtualized data centers to build private, public and hybrid IaaS (Infrastructure as a Service) clouds in existing infrastructures. It orchestrates storage, network, virtualization, monitoring, and security technologies to deploy VMs on distributed infrastructures [131]. As depicted in Fig. 6.1, each subscribed user or organization should have a VM for hosting the middleware and the needed servers such as storage and eventing servers. The Resource Allocator (RA) is responsible for receiving user subscribtion requests and resource allocation. Each VM should have a name, a DNS address and be accessible through the Internet to allow its owner to administrate it.

In order to add a new user to the system, i.e. a new VM, there are four different steps to be processed:

- **Template definitions** are defined and stored in a Template Repository by the administrator. Template defines a VM properties like the number of CPUs, CPU speed, memory, disks, the OS Image previously registered in the Image Repository, and configuration scripts. The OS Image is a linux server having pre-installed programs and iSensors middleware. There are different templates and OS Images to match each user needs. This step is done by the administrator only the first time or when adding or modifying some templates is needed.
- **Instanstiation :** Once a request to add a new user is received by the RA, it instanciates the specific template that matches the user needs using OpenNebula commands.
- **Virtual machine personalization :** The RA personalizes the VM : name, DNS, etc. then, it creates the new user account and adds user's public key to the authorized keys. Last, it returns to the new user needed information to access the VM, the middleware and servers.
- **User personalization :** user personalizes the middleware by instanciating composites and uploading needed scripts.

6.1.3 CRUD operations

The RA and Open Nebula make the run of CRUD operations possible on images, templates and VMs. CRUD stands for Create, Retrieve, Update and Delete operations. Operations on the virtual network and the way to connect a VM to the public network is strictly reserved to the platform administrator.

6.2 Social sensor networks

6.2.1 Motivations

Using the above solution, sensor data of different users are isolated in separated databases, middleware instances and virtual machines. In order to enable collaborations and sensor data sharing between users, we propose a social networking extension for the iSensors middleware which has many benefits:

- share sensor data.
- use gadgets (default/personalized) in composites.

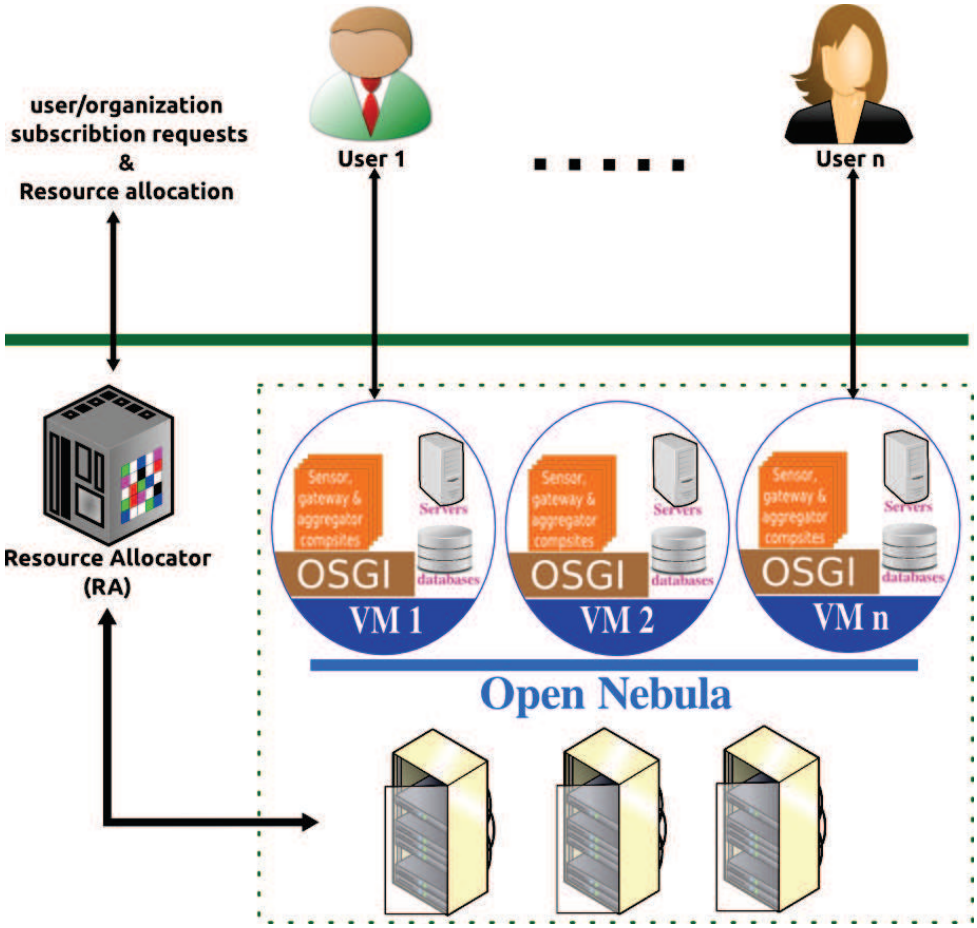


Figure 6.1: Global architecture of the cloud infrastructure for sensor networks.

- real-time monitoring.
- different security mechanisms.

6.2.2 Architecture

This extension can be seen as simply as adding gadgets to composites for providing user interfaces to visualize real-time and history sensor data using tables or graphs. Each user should have a personal portal, i.e. web and social mashup engine, where his own or other users gadgets can be imported. Each composite can be extended with a gadget to visualize data coming from sensors. A default gadget is provided in order to display, in a real-time manner, sensor data coming from a default topic in the eventing server. Personalized gadgets can be added by the user to provide more complex data visualization interfaces and to send commands to physical sensors.

As described in Fig. 6.2, the Social Sensor Network Infrastructure is composed of several parts:

- **The sensor Social Portal** is a web and social mashup engine that serves and hosts

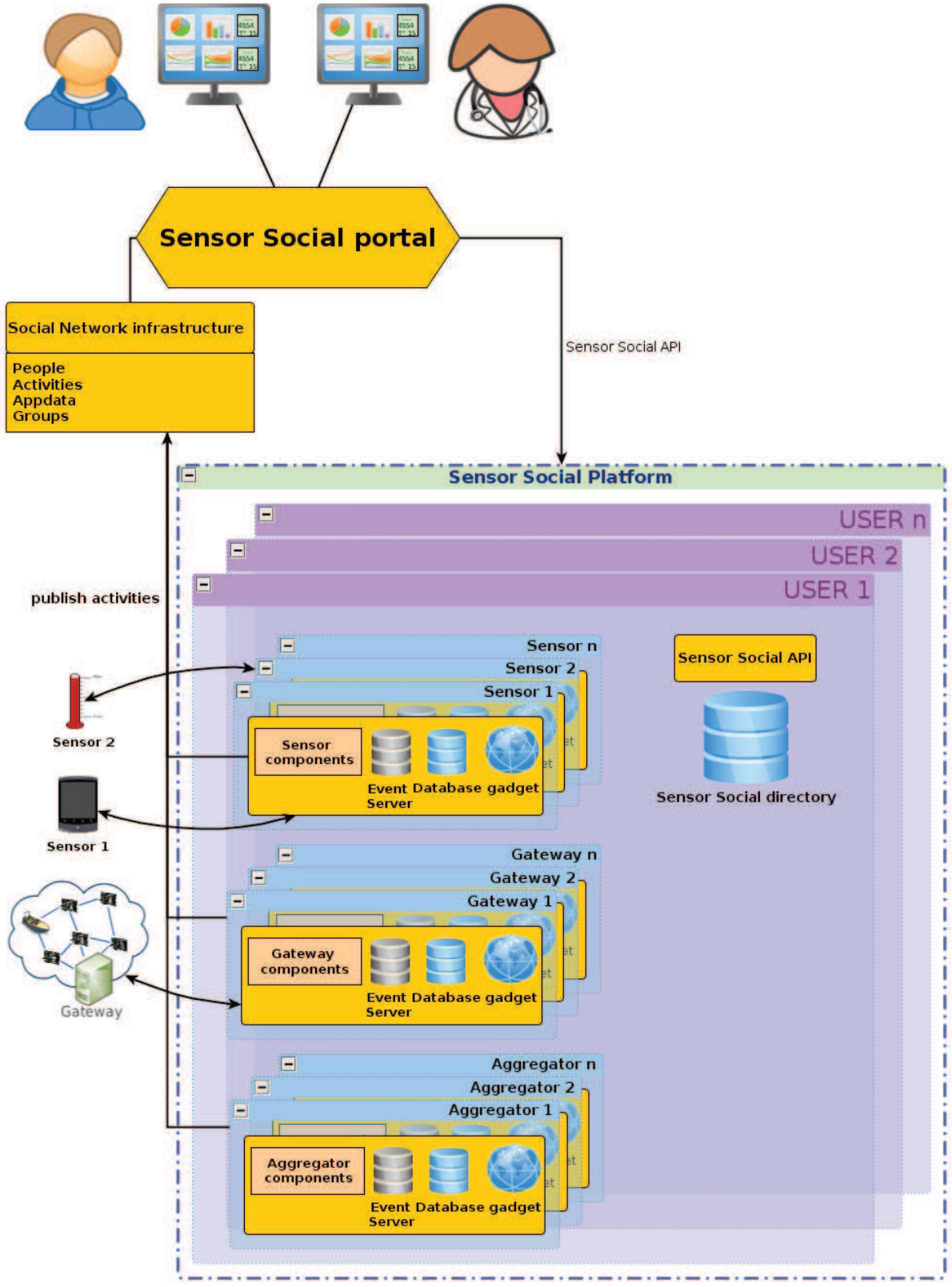


Figure 6.2: Global architecture of the Social Sensor Network Infrastructure.

social gadgets. It is hosted in a dedicated server. Each user have his own portal and can add, modify or remove personal/others sensor gadgets to organize his own page. Both the portal state and the page context are saved, so that at each connection, user finds the same interface.

- **The Social Network Infrastructure** contains the standard infrastructure of Social Networks and provides the Open Social services: people, activities, appdata and groups. Moreover, it stores all the information about users, their profiles, VMs relations and recent activities.

- **The Sensor Social Platform** is an extended version of the Sensor Cloud Infrastructure. Accordingly, each VM should contain in addition to default software, e.g. the iSensors middleware, storage and eventing servers, a new server named Sensor Social Directory and a Sensor Social API. Furthermore, a gadget is setup in each composite. New features of the Sensor Social Platform can be more clarified as follows:

  - *The Sensor Social API* is an extension to the Open Social API in order to standardize and facilitate the access from the portal website and gadgets to the Sensor Social Platform. So, accessing VMs, exploring users gadgets and navigating through sensor data is more intuitive.
  - *The Sensor Social Directory* is in each user VM to store information about his personal gadgets. These gadgets can be explored by the user himself or by other users with the intention of adding them to one's portal if he has sufficient authorizations. The list of all external users of gadgets is saved for the purpose of making the owner able to control the access to its gadgets.
  - *The Sensor Social Security* Security is an important issue in the Social Sensor Network Infrastructure. Each user should be authenticated to access the infrastructure using common used solutions like OAuth or public/private key. We assume in this proof of concept of the Social Sensor Network Infrastructure that the problem of users authentication can be easily solved. While, an attribute-based scheme is proposed in the next section to solve authorization issues.

6.3 Secure access for the data in the social sensor networks

As presented above, the social sensor network enables data sharing between users. As a result, one's friends see all his associated data which includes private and personal

data depending on the kind of sensors. These statements will not motivate people and organizations for using such a system unless a data protection mechanism is included in order to let users easily define authorizations and accesses.

6.3.1 Background

Access Control mechanisms are mandatory in social networks to protect users privacy. A very large set of solutions have been proposed in the literature that combines authentication and authorization mechanisms to access resources. Solutions can be grouped into five main classes.

The first class holds the simplest mechanism, where resources are simply protected by a ***username-password*** pair. This mechnism is very simple, but is limited as not all users need to have access to all resources.

The second class is ***Access Control Lists*** (***ACLs***), where each resource to which access should be controlled has its own associated list of mappings between the set of users requesting access to the resources and the set of actions that each user can take on the resources. ACLs are widely used, but the administration of the mappings is a cumbersome task when a large number of resources and users is involved, due to the fact that the ACL treats every user as a distinct entity with distinct sets of permissions for each resource.

The third class, the ***Role Based Access Control*** (***RBAC***), comes to solve this issue. In RBAC, the user's function or role determines whether access is granted or denied. Thus, access control is done using rules that define which roles can execute which actions on which resources. In addition, the user is allowed to be a member of multiple groups. Affecting users on roles can make the definition of granular access control a difficult task as a user can fall into a particular role, but do not need to have the full rights accorded to the other members of a group.

The fourth class is the ***Attribute-Based Access Control*** (***ABAC***) model where the access is controlled based on a set of characteristics, or attributes, associated with the user and/or the resource itself. Each attribute is a discrete, distinct field and can be a user's role. Access is granted when user's attributes fulfil the required combination of attributes. As can be seen, this model does not require to know the identity of the receiver as the data protection is using attributes. The issue of this model is that it can exists disparate attributes and access control mechanisms in the same organization.

The fifth class, the ***Policy-Based Access Control*** (***PBAC***) model, is an emerging model and an evolution of ABAC that tries to provide a more uniform access control model throughout the system.

CP-ABE : Ciphertext-Policy Attribute-Based Encryption

ABE extends the concept of Identity-Based Encryption (IBE) which was first introduced by Adi Shamir in 1984 [121] and provides an ABAC based on encryption. The first CP-ABE scheme was introduced in [132] where attributes are used to describe a user's credentials and the party encrypting data determines a policy for who can decrypt using an access structure. So, a user private key is associated with an arbitrary number of attributes while the access structure is described as a monotonic *access tree* where nodes are composed with threshold gates and leaves describe attributes. In fact, the structure `("parent" OR ("doctor" AND "Antoine-Béclère Hospital"))` can be defined to grant access for parent and doctors from Antoine-Béclère Hospital.

The CP-ABE scheme consists of four fundamental algorithms: Setup, Encrypt, Key Generation and Decrypt. In addition, there is an optional fifth algorithm: Delegate.

- ***Setup :*** The setup algorithm takes no input other than the implicit security parameter. It outputs public parameter PK and master key MK.
- ***Encrypt(PK,*** M***,*** $\mathbb{A}$***) :*** The algorithm takes as input public parameter PK, message M, and access structure $\mathbb{A}$ over the universe of attributes. It encrypts M and produce a ciphertext CT such that only a user that possesses the set of attributes that satisfies the access structure will be able to decrypt CT.
- ***Key Generation(MK,*** S***) :*** The algorithm is a function that takes as input a master key MK and the set of attributes S that describes the key. The output is a private key SK.
- ***Decrypt(PK,*** CT***,SK) :*** The inputs of the algorithm are public parameters PK, ciphertext CT, which contains access policy $\mathbb{A}$, and private key SK, which is the private key for the set of attributes S. If set S satisfies access structure $\mathbb{A}$ then the algorithm can decrypt the ciphertext and return a message M. Otherwise it returns an error.
- ***Delegate(SK,*** $\tilde{S}$***) :*** The inputs of the algorithm are secret key SK for some set of attributes S and set $\tilde{S} \subseteq S$. It outputs secret key $\tilde{SK}$ for the set of attributes $\tilde{S}$.

6.3.2 Related work

Many works in the litterature are using ABE to secure the access to data. The close three solutions to our work are described in this section.

In [133], Persona, an online social network (OSN) with user-defined privacy, was proposed. Persona hides user data with ABE, allowing users, not OSN, to apply fine-grained policies over who may view their data. Each user can assign his friends to groups. At

the beginning, each user generates an ABE public key and an ABE master secret key. Then, the user can generate for each friend an ABE secret key corresponding to the set of attributes that defines the groups that friend should be part of. A user has two objects: abstract resources and user data. Resources are protected using Access Control List (ACL) and user data is stored encrypted and can only be decryped by users belonging to the group for which the data was encrypted.

In [134], authors propose a system to provide a secure access for electronic health records (EHRs) in the cloud. There are three parties involved in this system: healthcare providers, an attribute authority (AA) and a cloud-based EHR system. EHRs are encrypted using CP-ABE and a policy based on authorized healthcare provider's attributes; and to decrypt EHRs, a healthcare provider must possess the set of attributes needed for proper access. EHRs are stored encrypted in the cloud-based EHR system. At initialization, AA generates the public key and the master private key. When a new healthcare provider joins the system, AA derives a distinct secret key associated with its attributes. Healthcare providers must regenerate keys after a predefined expiration date. An issue of this solution is that the patient is not considered as a system user.

Another system for the secure management of EHRs was proposed in [135]. Authors describe a variant of a CP-ABE scheme where the patient can encrypt its health records according to an access policy which has attributes issued by two trusted authorities TA_1 and TA_2. TA_1 authenticates users of the professional domain, and issue secret keys based on their attributes. The patient, who can take the role of the trusted authority (TA_2) of the social domain, might use the reputation of the users of the social domain to generate appropriate secret keys. For example, the patient can encrypt its health records such that only a user, who has the attribute `"Doctor"` issued from TA_1 or the attribute `"friend"` issued by TA_2, can decrypt it. Thus, the data is encrypted according to an access policy P=P_1 OR P_2 where P_1 is intended for the professional domain and P_2 is intended for the social domain. In addition, each measurement data has three related meta-data: a number (MD), a category (DC) and an administrator category (AC). Consequently, P_2 has this structure : P_2=a_{MD} OR a_{DC} OR a_{AC}, where a_x is an attribute of category x. This data categorization is intended to give the patient the choice of issuing keys to decrypt all or part of the measurements. A shared issue between those solutions is the fact of storing the encrypted form of data, which constitute a problem for the revocation and update of keys.

6.3.3 Our solution

This section proposes a security scheme to protect user sensor data and let him apply fine-grained policies over who may view his sensor data. In this scheme, data are still stored clear for two reasons. The first one is for allowing components (Alert generator,

Transformer, etc.) to analyse and transform data. The second reason is for allowing keys revocation and easy update without the need to decrypt all the stored data and then to encrypt them using new keys. Let assume the existence of a mechanism to protect access to the user's VM from outsiders and the administrator of the cloud infrastructure. Data encryption is done when data is going out or may go out the isensors middleware. In brief, data encryption is done when data is encapsulated at alerts sent through the eventing server or when data is sent as a response to a query received through the communication API. Note that data is also protected from the aggregator component which should have rights to decrypt them due to external users that can subscribe to topics in the eventing server.

CP-ABE was selected to encrypt data and protect the access to this data for many reasons. CP-ABE has many advantages as discussed in Sec. 6.3.1. In addition, it facilitates the management of access rights due to the use of attributes. When data is encrypted, it is associated with a combination of a set of attributes (access structure). Thus, only users having a private key that fulfil this required combination can decrypt the data. This is very useful as the composite does not need to know the identity of the receiver when generating alerts, so it encrypts data using attributes and a unique key instead of encrypting using each receiver key or a group key. Similarly, when data is displayed in the gadget included in the user's friend portal.

6.3.3.1 System architecture

To get access to the social sensor portal, the user must be authenticated using either a login/password pair or a public/private key. Thus, each user is uniquely identifiable. The user have friends and is the friend of other users; he can choose his friends and can request to be a friend of an other user, but he can not unless this latter approve the request. These functions are provided, in general, by the social network infrastructure.

In order to protect his data, the user issue a distinct private key associated with a set of attributes to each of his friends. Attributes can be roles, groups, affiliations and so on. When alerts or responses to queries are generated, they are encrypted using the public key of the user and an access policy which is a combination of attributes, only users (e.g. friends) having the required combination of attributes in their private keys can decrypt.

To provide these features, the security scheme need two additional parties in the architecture of the social sensor portal described in Sec. 6.2.2:

- ***Authorization Authority (AA) :*** it is unique and private for each *user*. It can be integrated in the Sensor Social Directory. AA is responsible of the generation and delivery of friends private keys and stores the user private keys. AA provides an interface for the user to assign attributes to friends, and to generate, update and revoke their private keys. When user U has a new friend F, the U's AA is

triggered to let U assign some attributes to F and generate the private key of F. Moreover, AA stores the private keys of user U delivered by his friends. These keys can be used by other components (for example the aggregator component) and are downloaded by U to use them in his browser.

- ***Local Security Component (LSC) :*** it is unique and private for each *composite*. First, it is responsible of encrypting alerts using the user's public key and an access policy. So, an additional element is added to the alert scheme which is `"access policy"`. This field is filled at the time of alert generation in the Alert Generator component. When an alert is received by the events manager, it is forwarded to LSC for applying the access policy and then returned encrypted to the events manager to proceed in its distribution. Second, it protects the access to GET queries through Easy and Advanced APIs in the communication API component using ACLs which are defined as lines of two columns. Each line contains the URL to be protected and the access policy.

6.3.3.2 Key management

Key management is a tricky part which ensures the proper execution of the access management protocol and should be cost-effective.

Generation and distribution : At the begining and during *VM instanstiation* or *VM personalization*, the ***Setup*** algorithm of the CP-ABE is executed. Then, when the user has a new friend, he assigns him some attributes, generates a private key and sends it to this new friend (or to his AA). In the other side, the user must receive her private key to access his friend data. Note that the user should have a key with all the set of attributes to get access to all his sensor data.

Revocation : To facilitate the management of the key life cycle, each attribute is tagged with a version like `doctor-v5.1` where "5" and "1" denote major and minor versions of the attribute respectively. Whenever the user wants to remove the right of a friend to use an attribute that he was authorized to use, he updates the minor version of the attribute and distributes new keys to his friends who keeps this attribute. After a certain time slot τ_a, whithout a minor or major version update of the attribute a, AA should update the major version of a and redistribute the keys to friends having the right to use a.

6.4 Conclusion

The sensor social portal proposed in this chapter facilitates the integration of Sensor networks into Social Networking Sites and also provides a Sensor Social Platform to setup

virtual composites that communicate with physical sensors and gateways or aggregate data. These composites can be added by the user on instantiating some default components or on uploading other manufacturer components (like drivers) for example. It also extends the iSensors middleware, defined in Chap. 3, and makes it ready to be provided for many users and organizations using cloud computing. In addition, the discussed solution offers a security scheme to protect the access to user's sensor data.

Chapter 7

Conclusions

Applications in WSNs are numerous and are spread over different areas where the network is mostly specific to the application and uses heterogeneous SNs as well as communication systems. In addition, the proliferation of sensors make each person or organization possesses heterogeneous sensors. Thus, data management complexity increases when considering WSN properties. This thesis has handled this complexity and presented new approaches for the collection, management and sharing of sensor data. Sec 7.1 presents a series of summaries of the achievements for each chapter of the dissertation. Then, we propose further research directions in Sec 7.2.

7.1 Summary of the dissertation

Chap. 2 summarized different aspects related to WSNs and the related background to our work. In the first part, the characteristics and limitations of the physical node has been highlighted as well as their capacity to form a WSN and the impact of their capabilities on these networks. Next, some applications of WSNs were enumerated to argue their importance and widespread use in various fields. Then, important properties of OSs were highlighted and TinyOS were presented as a sample. The second part was devoted to discuss some adressed problematics in this dissertation while presenting the most important related solutions in the literature. The first was sensor data management in WSNs. While the second and the third were concerned with the convergence between these networks and, respectively, cloud computing and social networking. Opportunities and challenges of these issues were discussed to highlight their importance to revolutionize sensor networks.

Chap. 3 proposed a middleware managing the reception, the storage, the indexing, the aggregation of sensor data from heterogeneous sensor nodes and the generation of alerts. We were interested to produce a middleware storing and processing data externally, unlike tinyDB [72], in order to have a better response time due to the use of a high performance

server where data is stored and indexed instead of collecting them from the network. Moreover, il is flexible because it does not impose a specific format of data or a specific program to load on sensor nodes unlike SWE [76] and QuadraSpace [75]. So, it is easily interconnected to existing WSNs without the need to update applications. Furthermore, we provide a RESTful API and alert server to allow external applications to interface easily and offer other additional services using the stored data and the generated alerts by the middleware.

In Chap. 4, we developed a system for monitoring water quality using the middleware presented in Chap. 3 and we demonstrated the possibility of its interfacing to an external application for data visualization. The system has been used in real-world experiments and its performance has been proven in test campaigns. We imposed the property of displaying data in a real-time manner for users which was not sufficiently investigated in the literature. We used cloud storage and no-SQL indexing solutions to offer an unlimited persistent and reliable storage and indexing systems in order to keep data for the long term.

Chap. 5 proposed a security scheme to secure the communication between the middleware and physical nodes, e.g. sensor and gateway nodes. It was applied in the context of Wireless Body Area Network (WBAN). After the analysis of the security scheme defined in [115], and discussed in Sec. 5.4, we found some issues such as the easy injection of false data and the inability to analyze data on the SS. Moreover, we identified the different properties and security level needs for each entity. Based on these observations, we proposed a hybrid authentication and key establishment scheme which is performed in two phases. The first one ensures the authentication and key establishment between the two entities : a gateway and the middleware. These entities need a high-security level as they communicate through the Internet and have enough resources to execute Identity-Based Cryptography (IBC) functions. Initially, the gateway generates a public key using IBC. During the authentication process, a pairwise key is established between the two entities based on the C-DH problem and the gateway gets a private key generated by the middleware. After the success of this phase, each sensor node can initiate the second phase in order to authenticate themselves to the gateway and the middleware. Also, the middleware authenticates itself and approves the gateway authenticity to the sensor node in order to reduce the computation load in the latter. During this phase, SN can only verify IBC signatures, and at its end, a three-party key is established and two pairwise keys are shared between SN and both the gateway and the middleware respectively. Thus the hybrid property is used to highlight the fact that asymmetric keys are generated and used during the first phase while only symmetric keys are generated and used in the second one due to the limited resources of SNs.

In Chap. 6, we realized that sharing sensor data among the users middleware to allow their collaboration and knowledge sharing was very challenging. For that reason, we

presented a secure social sensor portal for sharing sensor data between users. It is not integrated in other social sites like the proposed works in [90, 136]. Instead, it enables each sensor composite to provide its own generic or customized mash-up to be integrated in the portal unlike the other systems that offer a separate visualization system like those defined in Chap. 4 and [94]. In order to realise this feature, we hosted the middleware defined in Chap. 3 in a cloud infrastructure for sensor networks to make it available for many users as a softaware as a service (SaaS) where they have their private system to manage their sensor data. Under those circumstances, securing access to user data became a critical need to protect his privacy. For that reason, we propose a security scheme based on Attribute-Based Encryption [132] to protect user sensor data and let him apply fine-grained policies over who may view his sensor data. Moreover, we proposed a practical mechanism to manage key life cycle such as generation, distribution and revocation.

To summarize, in this dissertation we covered the secure data collection from sensor networks, its management and secure sharing in a multi-user SaaS.

7.2 Perspectives

The proposed middleware in Chap. 3 allows adding new composites which need to be configured when data must be translated or alerts must be generated. Moreover, the installation of a composite is not an easy task, despite its simplicity, for an ordinary user. For that reason, we proposed the market extension in Sec. 3.4.2. This remains limited as the user must seek the proper composite. Thus, adding an automatic discovery service in the middleware to discover the user's sensor nodes and download the proper components from the market to interface with them.

Apart from adding new composites, it is important to send commands to the sensor node from the middleware, for example after an alert to reconfigure or calibrate. Moreover, the traffic between them is not controlled by the middleware, so the push frequency is not optimized in the sensor node unless a specific algorithm is running. In order to handle these issues, a specific VM can be deployed on sensor nodes to create a communication tunnel between the SN and the middleware. Thus, the middleware can control the traffic frequency, execute standardized commands on the sensor node and implements the security scheme as defined in Chap. 5.

As a result, the formal validation of this security scheme, its implementation and performance measurements are very important to validate the theoretical results already proved in Chap. 5.

As a final but equally important note, finishing the implementation of the social sensor network proposed in Chap. 6 is very interesting to test it and validate the design. Although, some features need to be inverstigated. For example, how Bob can grant access for Alice to the data of his friend Oscar who is not a friend of Alice. But, Oscar should

be able to revoke Bob access and all of his granted friends or to refuse this grant. Using ABE, Bob can grant access to Alice without any need of the acceptance of Oscar.

Appendix A

Isfet case study

ISFET [112] were used during the mobesens project. They measures the water pH and temperature. The captured sensor data is sent to the back-end system in this format :

```
timeStamp;status;probeType;Vs;pH;Vt;temperature;pid
```

A sample of a sensor data :

```
1339599780;0;pH;1470;11.40;1870.6;13.8;2
```

```
xquery version "1.0";

(: $Id: devguide_xquery.xml 15318 2011-09-07 19:43:24Z dizzzz application/xquery $ :)

import module namespace request="http://exist-db.org/xquery/request";
import module namespace session="http://exist-db.org/xquery/session";
import module namespace util="http://exist-db.org/xquery/util";
import module namespace  fn="http://www.w3.org/2005/xpath-functions";

declare option exist:serialize "method=xhtml␣media-type=text/xml";

declare function local:transformerIsfet($sequence as item()*) as element()*
{
    let $count := count($sequence)
    let $seqelem := ('timeStamp', 'status', 'probeType', 'Vs', 'pH', 'Vt', 'temperature',
         'pid')

    return
     if ($count = 8) then (: Verify fields number of ISFET Data :)
  let $res1 := for $item at $id in $sequence
            return
             <metadata attribute='{$seqelem[$id]}'>{$item}</metadata>
           return local:validateIsfet($res1, $count + 1)
        else
         <metadata attribute='error'>measurement length {$count} is not the needed length
              </metadata>

};

declare function local:validateIsfet($seq as element()*, $pos as xs:integer) as element()
    *
{
 let $valiAttr := 'validation'
 let $validation_status := ('PreValidated', 'YES', 'NOT')

 return if (fn:number($seq[@attribute="status"]) != 0 or (fn:number($seq[@attribute="Vt"
     ]) > 4000) or ((fn:number($seq[@attribute="Vt"]) > 3200) and (fn:number($seq[
     @attribute="Vs"]/text()) > 3500))) then
  insert-before($seq, $pos , (<metadata attribute='{$valiAttr}'>{$validation_status[3]}</
      metadata>))
 else
  insert-before($seq, $pos , (<metadata attribute='{$valiAttr}'>{$validation_status[1]}</
      metadata>))
};

declare function local:main() as element()*
{
 (: recuperation des parametres du request :)
 let $nodeID := request:get-parameter("nodeID", "")
 let $memory := request:get-parameter("memory", "")
 let $data_id := request:get-parameter("data_id", "")
 let $configID := request:get-parameter("configID", "")

 (: recuperation des donnees :)
 let $binary := util:binary-doc($data_id)
 let $input := util:binary-to-string($binary)

 let $delimeter := if (not ($configID))
                   then ';'
                   else ';'

 let $sequence := tokenize($input, $delimeter)

 return  local:transformerIsfet($sequence)
};

<metalist>

{local:main()}

</metalist>
```

Listing A.1: Translation script of ISFET data.

```
xquery version "1.0";

(: $Id: Tue 19 Jun 2012 01:53:53 PM CEST -- application/xquery $ :)

import module namespace request="http://exist-db.org/xquery/request";
import module namespace session="http://exist-db.org/xquery/session";
import module namespace util="http://exist-db.org/xquery/util";
import module namespace  fn="http://www.w3.org/2005/xpath-functions";

declare option exist:serialize "method=xhtml␣media-type=text/xml";

(: CONSTANT definitions :)
declare variable $seq_types := ('jms', 'rest', 'rss+atom', 'xmpp', 'email', 'sms');
declare variable $seq_contentTypes := ('Text/xml', 'application/atom+xml', 'application/
    json', 'text/plain');

(: recuperation des parametres du request :)
declare variable $nodeID := request:get-parameter("nodeID", "");
declare variable $memory := request:get-parameter("memory", "");
declare variable $data_id := request:get-parameter("data_id", "");
declare variable $configID := request:get-parameter("configID", "");

(: recuperation des donnees :)
declare variable $binary := util:binary-doc($data_id);
declare variable $input := util:binary-to-string($binary);
declare variable $input_xml := util:parse($input);

(: declare variable $input_xml := doc($data_id); :)

declare function local:getFireAlert() as element()*
{
 (: trigger fire alert  :)
 if ((fn:number($input_xml//metadata[@attribute="temperature"]) > 60) and (($input_xml//
     metadata[@attribute="validation"]/text() = 'PreValidated') or ($input_xml//metadata[
     @attribute="validation"]/text() = 'YES'))) then
   (
   <alert>
    <type>{$seq_types[1]}</type>
    <content-type>{$seq_contentTypes[4]}</content-type>
    <destination>{$nodeID}-alerts</destination>
    <msg>Attention!! ***Fire*** !! Temperature value is {$input_xml//metadata[@attribute=
        "temperature"]//text()}. </msg>
   </alert>
   )
 else
   ()
};

declare function local:getNewDataAlert() as element()*
{
 (
   <alert>
    <type>{$seq_types[1]}</type>
    <content-type>{$seq_contentTypes[1]}</content-type>
    <destination>{$nodeID}-updates</destination>
    <msg>{$input_xml}</msg>
   </alert>
  )
};

declare function local:main() as element()*
{
(
 <memory>{$data_id}</memory>,
 <alerts>
  {local:getFireAlert()}
  {local:getNewDataAlert()}
 </alerts>
)
};


<GeneratedAlerts>
 {local:main()}
</GeneratedAlerts>
```

Listing A.2: Alert generation script to be executed on ISFET data.

Appendix B

Ebro river campaign

B.1 Objectives

The Ebro river campaign was an opportunity to test improvements to the visualisation interfaces of the back-end system and the new APIs, especially the interface to the KML fusion file that links GPS coordinates to the sensor data: at the campaign, the coordinates and the measurements are achieved by different means and devices to allow reliable localization of the monitored sites. In this test, the back-end system has the key role of ensuring monitoring of sensors and networking is supported and provided, following deployment on a 24 hour basis. The back-end system offers analysis and measurements validation services as well.

B.2 System description

As described in Sec.4.3, the gateway PC installed inside the Xerta station was connected to a GSM/GPRS router in order to send the sensor collected data from the SinkNodes to the back-end system. During the Ebro river test, data were sent to the back-end system according either in real time or opportunistically in bursts when node connectivity to the back-end becomes available.

B.3 Results

Since data sent from ISFET sensors can be corrupted or contain some errors, it was necessary to provide a validation service from the back-end system to administrators reinforced by a pre-validation service due to the massive amounts of data received from the campaigns. The pre-validation consists of tagging measurements with a flag in the meta-data associated to a measurement. This flag indicates the status of the measurement prior to analysis by an operator for final validation via the graphical user interface of

the back-end visualisation system. The pre-validation status flags are: "pre-validated", "yes" or "not". The algorithm developed for this purpose is depicted in Fig.B.1. The algorithm first verifies if the status flag indicates that there are no errors and if so checks the measurement value against a legitimate interval. If the measurement falls within the expected interval, the flag is set to the pre-validated status, otherwise it is declared as not pre-validated. The administrator, operator or scientific expert can now approve or disapprove the pre-validation by setting the flag to "yes" or "not". To improve the capacity of the administrator to monitor the status of ISFET sensors, the validation flag is presented in real-time to the expert via the visualisation system interface as depicted in Fig. B.2.

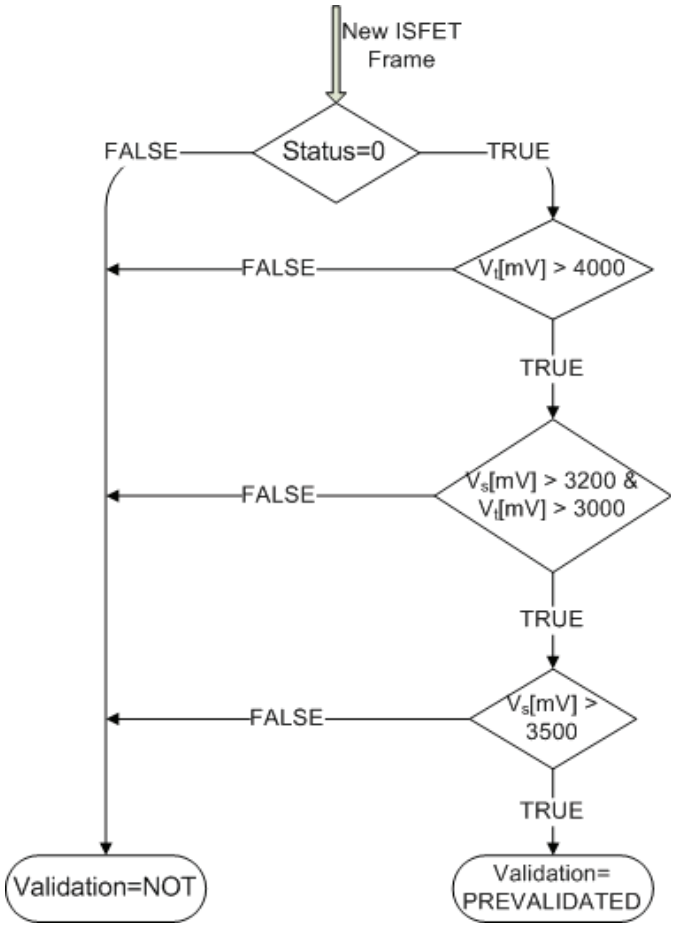


Figure B.1: Pre-validation algorithm.

The display of real-time ISFET data in a table is not very practical as it presents to the end user only the latest data. This information does not reflect the measurement status and dynamic changes. To improve visualisation of the data, a graph displaying in real time the measurements and the received data history was added to the interface. The graph display can be activated or disabled by the end user according to preferences and needs. This interface, depicted in Fig. B.2, enables real-time visualisation of the ISFET sensor data and thus remote detection of sensor failures.

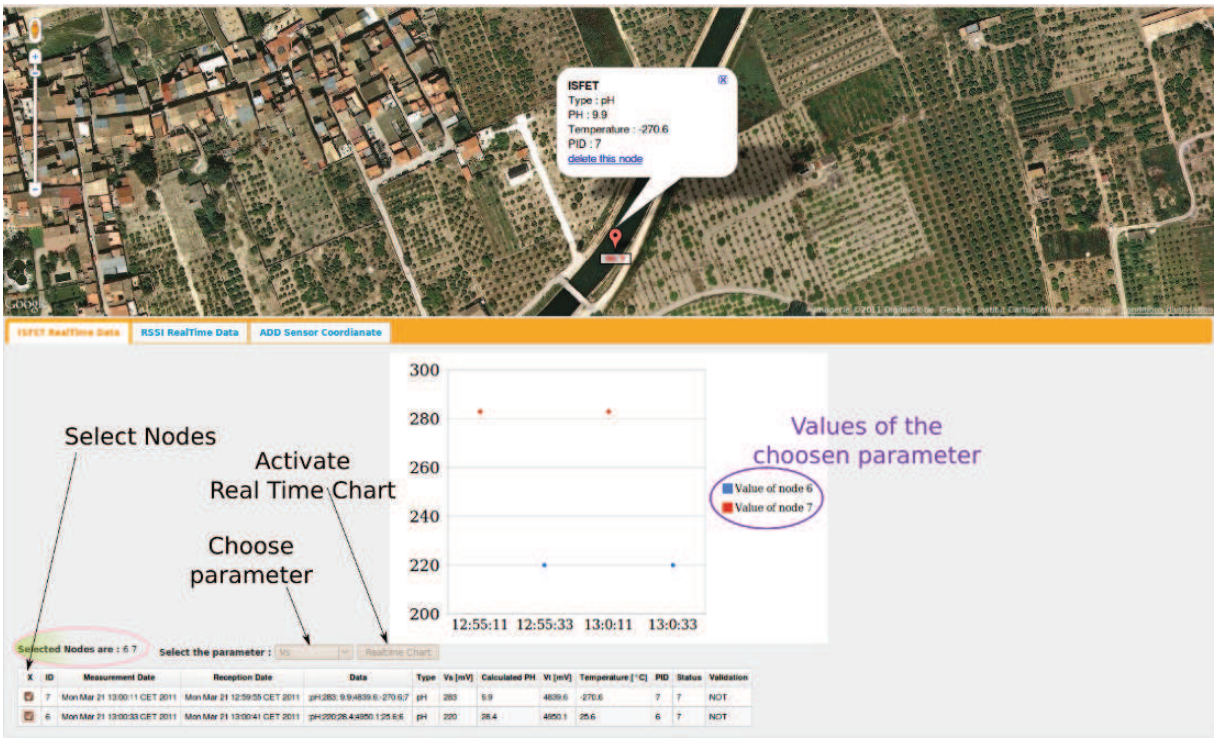


Figure B.2: Real-time visualization interface of ISFET data.

These tests were also the opportunity to remove some inconsistencies and inconvenient data representations. The visualization and interpretation of RSSI real time-data, expressed in hexadecimal values, make the representation in the real time tables inappropriate because of fast RSSI changes as each node has an associated value for each neighbour. A visual interpretation was preferred, developed and integrated in the interface to allow intuitive interpretation and understanding of the networking situation between the measurement nodes. A visualisation module was consequently added to the RSSI real time interface to draw colour coded links to indicate the RSSI values between nodes. Fig. B.3 illustrates how the real time RSSI data is presented in the visualisation graphical user interface.

Another feature was also added so that when users point the mouse on a link between two nodes, an information window is automatically prompted to display all information on the selected link as shown in Fig. B.4.

When clicking on the RSSI colour configuration icon (or button), a dialog window (see Fig. B.5) pops up to allow the user to configure themselves the associated colour for each RSSI value.

During these last tests, CSEM deployed autonomous GPS sensors along with the WiseNodes. These nodes do not send real time data to the back-end system but produce after the measurement campaign a KML file that contains GPS coordinates related to the localization of sites where measurements were conducted. There was therefore a

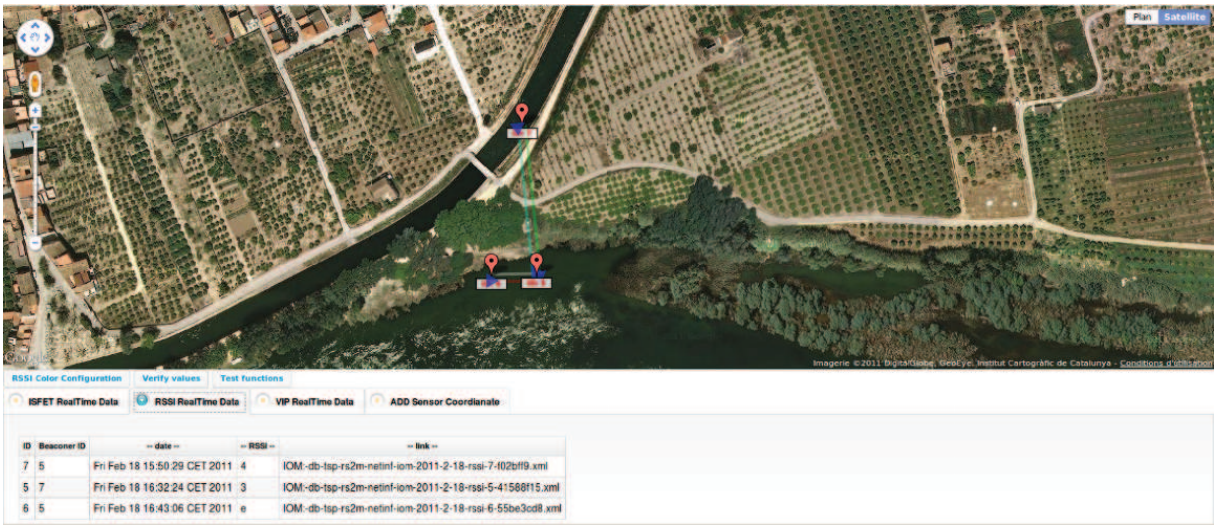


Figure B.3: RSSI visualization interface.

need to fuse the GPS data (provided as a KML file) with the measurement data coming separately from the ISFET sensor probes, to produce a richer KML file. The Fusion algorithm is described in Fig. B.6. Using the resulting rich and fused KML file, points where measurements were conducted are visualised using flags, and visited points are visualised using squares. Clicking on a flag displays an associated window to display the ISFET measurements made at the selected point.

As described before, the ISFET data were at first step validated automatically through an algorithm, enabling at a second step the administrator or expert to validate data manually through the visualization interface with the display of sensor history data. The user can set the validation status to "yes" or "not" (as highlighted in blue in Fig. B.7). The user can also export validate data in CSV or excel format as highlighted in red in Fig. B.7.

The back-end system provides also the opportunity during and after tests to visualise and to validate ISFET history data in tables as described in Fig. B.7. To ease the interpretation of data, an interface that generates dynamic graphs in the back-end system was available during tests to assist monitoring of ISFET measurement through the back-end visualisation and interpretation services. Fig. B.8 presents a screen shot of this interface. To generate the graphic, users select the day of the campaign, the ISFET identifier (PID), the measurement parameter (Vs, pH, Vt or temperature) and the validation state of the data (all states (field = all), pre-validated, validated (YES) or not valid (NO)).

Figure B.4: Info window representation when moving mouse on a line.

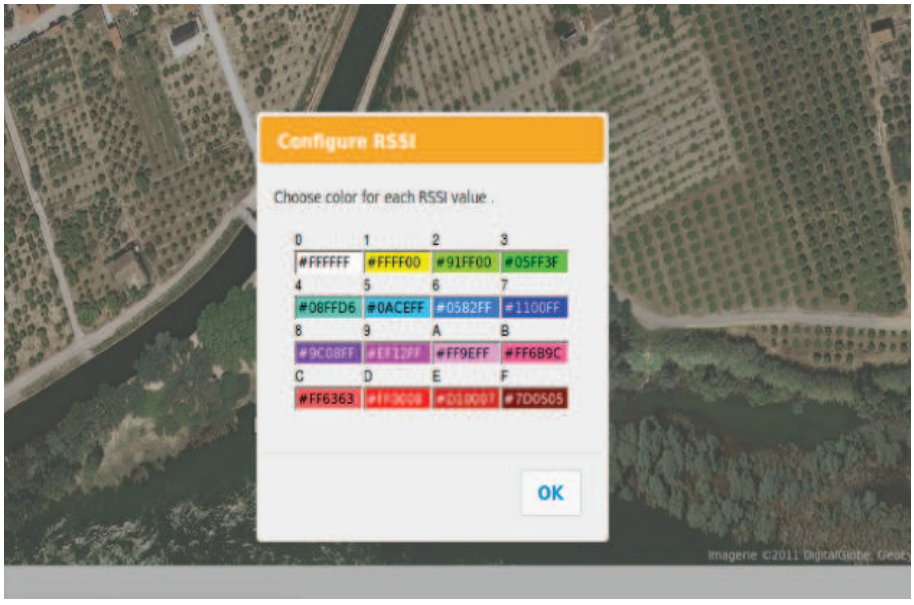


Figure B.5: Configuration of pairs: RSSI value and color.

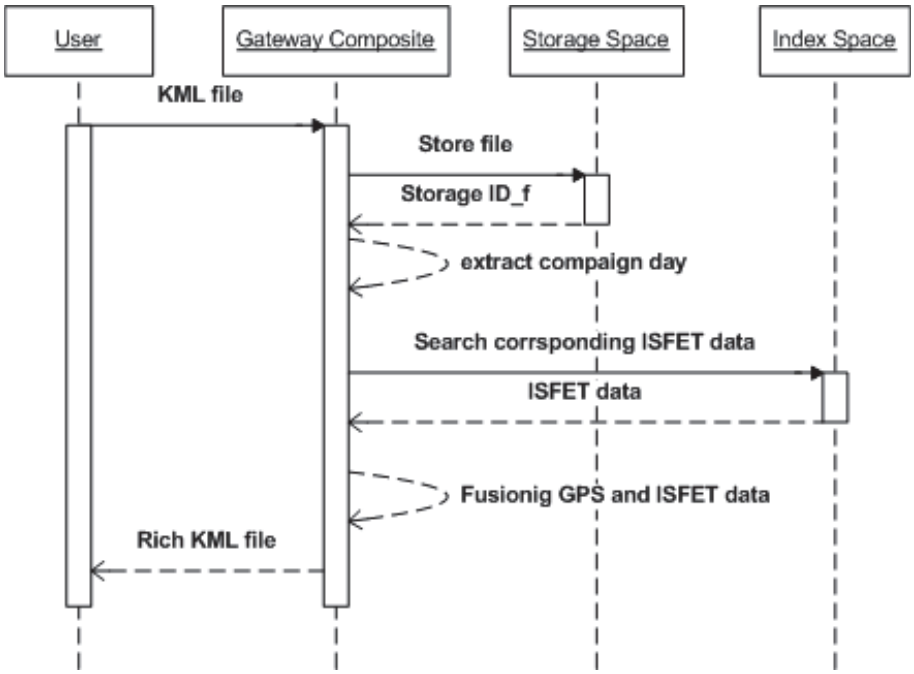


Figure B.6: Fusion between KML file and ISFET data.

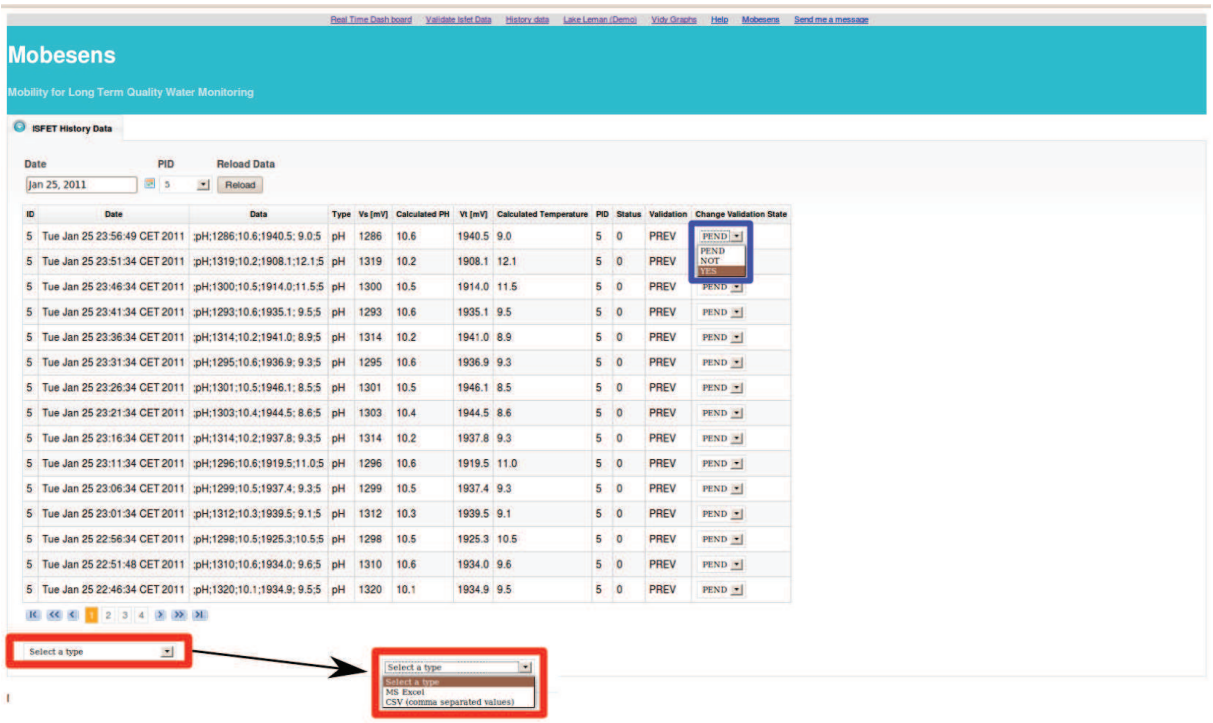


Figure B.7: Visualization and validation of ISFET history data.

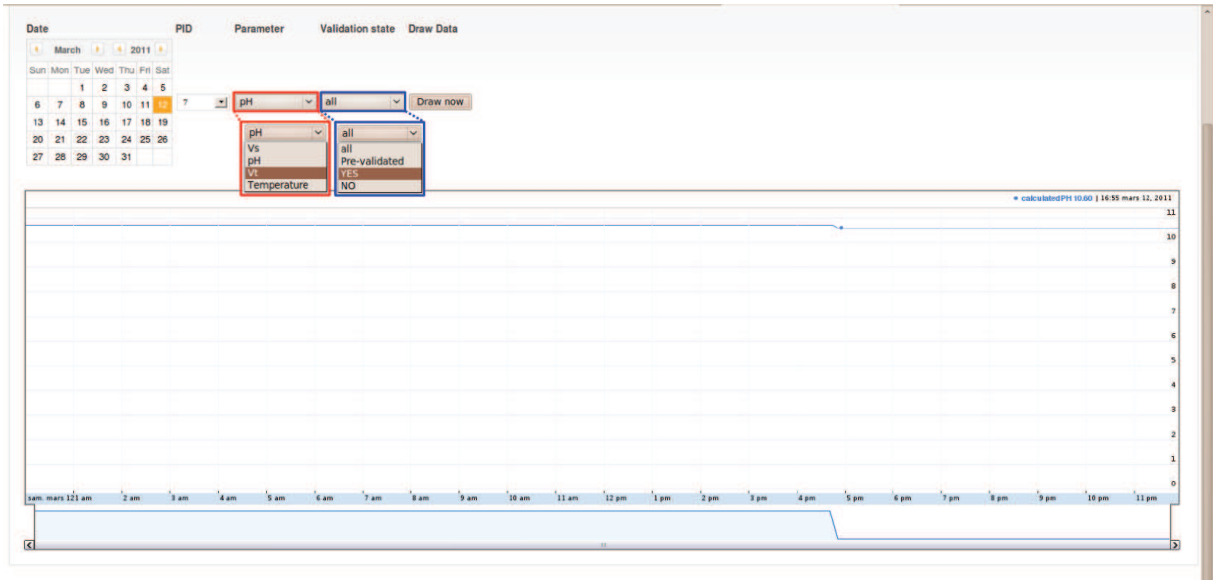


Figure B.8: ISFET history data visualization interface.

Bibliography

[1] Moteiv Corporation. Tmote sky : Datasheet, 2006.

[2] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Technical report, 2006.

[3] Crossbow. Micaz. http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf.

[4] Moteiv Corporation. Telosb: Datasheet. http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152%3Atelosb, 2004.

[5] BTnode rev3 Hardware Reference. http://www.btnode.ethz.ch/Documentation/BTnodeRev3HardwareReference.

[6] Sun SPOT. http://www.sunspotworld.com/docs/index.html.

[7] Intel Imote 2. http://embedded.seattle.intel-research.net/wiki/index.php?title=Intel_Mote_2.

[8] Y. Chen and Q. Zhao. On the lifetime of wireless sensor networks. *Communications Letters, IEEE*, 9(11):976 – 978, nov. 2005.

[9] S.K. Singh, M.P Singh, and D.K. Singh. Routing Protocols in Wireless Sensor Networks - A Survey. *International Journal of Computer Science & Engineering Survey*, 1(2):63–83, November 2010.

[10] J. Beutel, K. Römer, M. Ringwald, and M. Woehrle. Deployment techniques for sensor networks. In Gianluigi Ferrari, editor, *Sensor Networks*, Signals and Communication Technology, pages 219–248. Springer Berlin Heidelberg, 2009.

[11] C. Basaran and K.D. Kang. Quality of service in wireless sensor networks. In Subhas Chandra Misra, Isaac Woungang, and Sudip Misra, editors, *Guide to Wireless Sensor Networks*, Computer Communications and Networks, pages 305–321. Springer London, 2009.

[12] W. Drira, C. Bekara, and M. Laurent. Secure Aggregation in Wireless Sensor Networks. Research report 09-008-LOR, Télécom SudParis, Evry, july 2009.

[13] C.F. García-hernández, P.H. Ibargüengoytia-gonzález, J. García-hernández, and J.A. Pérez-díaz. Wireless sensor networks and applications: a survey. 2007.

[14] W.M. Merrill, F. Newberg, K. Sohrabi, W. Kaiser, and G. Pottie. Collaborative networking requirements for unattended ground sensor systems. In *Aerospace Conference, 2003. Proceedings. 2003 IEEE*, volume 5, pages 2153 – 2165, 8-15, 2003.

[15] M.P. Durisic, Z. Tafa, G. Dimic, and V. Milutinovic. A survey of military applications of wireless sensor networks. In *Embedded Computing (MECO), 2012 Mediterranean Conference on*, pages 196 –199, june 2012.

[16] M. Bahrepour, N. Meratnia, and P.J.M. Havinga. Automatic fire detection: A survey from wireless sensor network perspective, December 2008.

[17] E.A. Basha, S. Ravela, and D. Rus. Model-based monitoring for early warning flood detection. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 295–308, New York, NY, USA, 2008. ACM.

[18] L.W. Yeh, Y.C. Wang, and Y.C. Tseng. iPower: an energy conservation system for intelligent buildings by wireless sensor networks. *Int. J. Sen. Netw.*, 5(1):1–10, February 2009.

[19] L. Cheng and S.N. Pakzad. Agility of wireless sensor networks for earthquake monitoring of bridges. In *Proceedings of the 6th international conference on Networked sensing systems*, INSS'09, pages 212–215, Piscataway, NJ, USA, 2009. IEEE Press.

[20] S. Hussain, S. Schaffner, and D. Moseychuck. Applications of wireless sensor networks and rfid in a smart home environment. In *Proceedings of the 2009 Seventh Annual Communication Networks and Services Research Conference*, CNSR '09, pages 153–157, Washington, DC, USA, 2009. IEEE Computer Society.

[21] M. Erol-Kantarci and H.T. Mouftah. Wireless sensor networks for smart grid applications. In *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International*, pages 1 –6, april 2011.

[22] F. Stajano, N. Hoult, I. Wassell, P. Bennett, C. Middleton, and K. Soga. Smart bridges, smart tunnels: Transforming wireless sensor networks from research prototypes into robust engineering infrastructure. *Ad Hoc Networks*, 8(8):872 – 888, 2010.

[23] P. J. Bennett, Y. Kobayashi, K. Soga, and P. Wright. Wireless sensor network for monitoring transport tunnels. *Proceedings of the Institution of Civil Engineers - Geotechnical engineering*, 163(3):147–156, 2010. eng.

[24] A. Pantelopoulos and N. G. Bourbakis. A survey on wearable sensor-based systems for health monitoring and prognosis. 40(1):1–12, 2010.

[25] B. Latré, B. Braem, I. Moerman, C. Blondia, and P. Demeester. A survey on wireless body area networks. *Wirel. Netw.*, 17:1–18, January 2011.

[26] M. Becker, C. Bekara, A-L. Beylot, A. Delye, R. Dhaou, C. Diallo, W. Drira, F. Joseph, V. Gauthier, A. Gupta, R. Kacimi, I. Lanteri, M. Laurent, M. Marot, D. Périno, and E. Préterre. Cold chain monitoring using wireless sensor networks : design & experimental feedback. Rapport de recherche RR-09007-RST, Télécom Sud Paris, Evry, mai 2009.

[27] K. Martinez, J.K. Hart, and R. Ong. Environmental sensor networks. *Computer*, 37(8):50–56, August 2004.

[28] Y.J. Jung, Y.K. Lee, D.G. Lee, K.H. Ryu, and S. Nittel. Air pollution monitoring system based on geosensor network. In *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, volume 3, pages III –1370 –III –1373, july 2008.

[29] N.P. Preve and E.N. Protonotarios. An integrated sensor web grid cyberimplementation for environmental protection. *Sensors Journal, IEEE*, 11(9):1787 –1794, sept. 2011.

[30] Z. Wang, Q. Wang, and X. Hao. The design of the remote water quality monitoring system based on wsn. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, pages 1 –4, sept. 2009.

[31] N. Jin, R. Ma, Y. Lv, X. Lou, and Q. Wei. A novel design of water environment monitoring system based on wsn. In *Computer Design and Applications (ICCDA), 2010 International Conference on*, volume 2, pages V2–593 –V2–597, june 2010.

[32] S.A. Ruberg, R.W. Muzzi, S.B. Brandt, J.C. Lane, T.C. Miller, J.J. Gray, S.A. Constant, and E.J. Downing. A wireless internet-based observatory: The real-time

coastal observation network (recon). In *OCEANS 2007*, pages 1 –6, 29 2007-oct. 4 2007.

[33] M.A. Taleghan, A. Taherkordi, M. Sharifi, and T.H. Kim. A survey of system software for wireless sensor networks. In *Future Generation Communication and Networking (FGCN 2007)*, volume 2, pages 402 –407, dec. 2007.

[34] M.O. Farooq and T. Kunz. Operating systems for wireless sensor networks: A survey. *Sensors*, 11(6):5900–5930, 2011.

[35] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An Operating System for Sensor Networks Ambient Intelligence. In Werner Weber, Jan M. Rabaey, and Emile Aarts, editors, *Ambient Intelligence*, chapter 7, pages 115–148. Springer Berlin Heidelberg, Berlin/Heidelberg, 2005.

[36] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455 – 462, nov. 2004.

[37] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *MONET*, 10(4):563–579, 2005.

[38] A. Eswaran, A. Rowe, and R. Rajkumar. Nano-rk: an energy-aware resource-centric rtos for sensor networks. In *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, pages 10 pp. –265, dec. 2005.

[39] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 233–244, Washington, DC, USA, 2008. IEEE Computer Society.

[40] K. Klues, C-J.M. Liang, J. Paek, R. Musăloiu-E, P. Levis, A. Terzis, and R. Govindan. Tosthreads: thread-safe and non-invasive preemption in tinyos. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 127–140, New York, NY, USA, 2009. ACM.

[41] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. *SIGPLAN Not.*, 38(5):1–11, May 2003.

[42] J.N. Al-karaki and A.E. Kamal. Routing techniques in wireless sensor networks: A survey. *IEEE Wireless Communications*, 11:6–28, 2004.

[43] D. Goyal and M.R. Tripathy. Routing protocols in wireless sensor networks: A survey. In *Advanced Computing Communication Technologies (ACCT), 2012 Second International Conference on*, pages 474 –480, jan. 2012.

[44] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, HICSS '00, pages 8020–, Washington, DC, USA, 2000. IEEE Computer Society.

[45] A. Perrig, Rv Szewczyk, J.D. Tygar, V. Wen, and D.E. Culler. Spins: security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, September 2002.

[46] D. Braginsky and D. Estrin. Rumor routing algorthim for sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 22–31, New York, NY, USA, 2002. ACM.

[47] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31:2002, 2002.

[48] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *ACM MOBICOM*, pages 70–84, 2001.

[49] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical report, 2001.

[50] D. Niculescu and B. Nath. Trajectory based forwarding and its applications. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, MobiCom '03, pages 260–272, New York, NY, USA, 2003. ACM.

[51] S. Lindsey, C. Raghavendra, and K.M. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Trans. Parallel Distrib. Syst.*, 13(9):924–935, September 2002.

[52] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16, 2002.

[53] Tian He, J.A. Stankovic, Chenyang Lu, and T. Abdelzaher. Speed: a stateless protocol for real-time communication in sensor networks. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 46 – 55, may 2003.

[54] Bluetooth. https://www.bluetooth.org/Technical/Specifications/adopted.htm.

[55] J. Zhang, L. Shan, H. Hu, and Y. Yang. Mobile cellular networks and wireless sensor networks: toward convergence. *Communications Magazine, IEEE*, 50(3):164 –169, march 2012.

[56] Z. Wang, K. Yang, and D.K. Hunter. Modelling and analysis of convergence of wireless sensor network and passive optical network using queueing theory. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pages 37 –42, oct. 2011.

[57] J.W. Hui and D.E. Culler. Ipv6 in low-power wireless networks. *Proceedings of the IEEE*, 98(11):1865 –1878, 2010.

[58] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), August 2007.

[59] E. Kim, D. Kaspar, and JP. Vasseur. Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6568 (Informational), April 2012.

[60] E. Kim, D. Kaspar, C. Gomez, and C. Bormann. Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing. RFC 6606 (Informational), May 2012.

[61] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (Proposed Standard), September 2011.

[62] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), September 2007. Updated by RFC 6282.

[63] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet-Standard Management Framework. RFC 3410 (Informational), December 2002.

[64] J. Schoenwaelder, H. Mukhtar, S. Joo, and K. Kim. SNMP Optimizations for Constrained Devices. draft, 2011.

[65] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007. Updated by RFC 5942.

[66] Z. Shelby, S. Chakrabarti, and E. Nordmark. Neighbor Discovery Optimization for Low Power and Lossy Networks. draft-21, 2012.

[67] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim. Tinyrest - a protocol for integrating sensor networks into the internet. In *in Proc. of REALWSN*, 2005.

[68] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP). draft-12, October 2012.

[69] A.P. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby, and M. Zorzi. Architecture and protocols for the internet of things: A case study. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 678 –683, 29 2010-april 2 2010.

[70] D. Guinard. Towards the web of things: Web mashups for embedded devices. In *In MEM 2009 in Proceedings of WWW 2009. ACM*, 2009.

[71] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, March 2005.

[72] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal Ubiquitous Comput.*, 10(1):37–44, December 2005.

[73] Y. Xu, A. Helal, and M. Guizani. Cloud-edge-beneath architecture for energy-efficient cloud-sensor systems. *Wireless Communications, IEEE*, november 2012.

[74] G. Coulson, D. Kuo, and J. Brooke. Sensor networks + grid computing = a new challenge for the grid? *Distributed Systems Online, IEEE*, 7(12):2, dec. 2006.

[75] QuadraSpace Protocol. http://quadraspace.googlegroups.com/web/qs_protocol_DRAFT4.pdf, 2009.

[76] S. Ingo. OGC Sensor Web Enablement Architecture. Best Practice 06-021r4, Open Geospatial Consortium, Inc., 2008.

[77] C. Marin and M. Desertot. Sensor bean: a component platform for sensor-based services. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, MPAC '05, pages 1–8, New York, NY, USA, 2005. ACM.

[78] Cloud computing. http://en.wikipedia.org/wiki/Cloud_computing.

[79] M. Deng, M. Petkovic, M. Nalin, and I. Baroni. A home healthcare system in the cloud–addressing security and privacy challenges. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 549 –556, july 2011.

[80] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(5):85–95, October 2002.

[81] R. Barr, J.C. Bicket, D.S. Dantas, B. Du, T.W.D. Kim, B. Zhou, and E. Sirer. On the need for system-level support for ad hoc and sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(2):1–5, April 2002.

[82] Y. Yu, L.J. Rittle, V. Bhandari, and J.B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 139–152, New York, NY, USA, 2006. ACM.

[83] N. Raveendranathan, S. Galzarano, V. Loseu, R. Gravina, R. Giannantonio, M. Sgroi, R. Jafari, and G. Fortino. From modeling to implementation of virtual sensors in body sensor networks. *Sensors Journal, IEEE*, 12(3):583 –593, march 2012.

[84] M. Yuriyama and T. Kushida. Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing. In *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, pages 1 –8, sept. 2010.

[85] FRESNEL: Federated Secure Sensor Network Laboratory. http://www.cl.cam.ac.uk/research/srg/netos/fresnel/.

[86] SensEye - Multi-Tier, Multi-Modal Camera Sensor Network. http://sensors.cs.umass.edu/projects/senseye/.

[87] G. Fortino, A. Guerrieri, F. Bellifemine, and R. Giannantonio. Platform-independent development of collaborative wireless body sensor network applications: Spine2. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 3144 –3150, oct. 2009.

[88] M.M. Islam, M.M. Hassan, G.W. Lee, and E.N. Huh. A survey on virtualization of wireless sensor networks. *Sensors*, 12(2):2175–2207, 2012.

[89] d.m. boyd and N.B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1), 2007.

[90] A. Kamilaris and A. Pitsillides. Social networking of the smart home. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2010 IEEE 21st International Symposium on*, pages 2632 –2637, sept. 2010.

[91] M.A. Rahman, A. El Saddik, and W. Gueaieb. Senseface: A sensor network overlay for social networks. In *Instrumentation and Measurement Technology Conference, 2009. I2MTC '09. IEEE*, pages 1031 –1036, may 2009.

[92] M.A. Rahman, A. El Saddik, and W. Gueaieb. Building dynamic social network from sensory data feed. *Instrumentation and Measurement, IEEE Transactions on*, 59(5):1327 –1341, may 2010.

[93] M.A. Rahman, M.F. Alhamid, A. El Saddik, and W. Gueaieb. A framework to bridge social network and body sensor network: An e-health perspective. In *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, pages 1724 –1727, 28 2009-july 3 2009.

[94] B.L. Gorman, D.R. Resseguie, and C. Tomkins-Tinch. Sensorpedia: Information sharing across incompatible sensor systems. In *Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on*, pages 448 –454, may 2009.

[95] W. Drira, É. Renault, and D. Zeghlache. isensors: A middleware for dynamic sensor networks. In *Communications and Information Technology (ICCIT), 2012 International Conference on*, pages 134 –138, june 2012.

[96] W. Drira, É. Renault, and D. Zeghlache. Towards a dynamic middleware for wireless sensor networks. In *Networking and Future Internet (ICNFI), 2012 Second International Conference on*, pages 42 –45, april 2012. ISBN-13: 978-2-915618-24-2.

[97] eXist-db, the Open Source Native XML Database. `http://exist.sourceforge.net`.

[98] XMPP protocol. `http://xmpp.org/`.

[99] Java Message Service (JMS). `http://java.sun.com/products/jms`.

[100] OSGi alliance. `http://www.osgi.org/Main/HomePage`.

[101] C. Escoffier, R.S. Hall, and P. Lalanda. ipojo: an extensible service-oriented component framework. In *Services Computing, 2007. SCC 2007. IEEE International Conference on*, pages 474 –481, july 2007.

[102] karaf. `karaf.apache.org`.

[103] W. Drira, É. Renault, and D. Zeghlache. Design and performance evaluation of a composite-based back end system for wsns. In *International Wireless Communications and Mobile Computing Conference (IWCMC), 2012 IEEE 8th International Conference on*, august 2012. ISBN 978-1-4577-1379-8.

[104] W. Drira, É. Renault, and D. Zeghlache. Design and performance evaluation of a system for storing and visualizing data from a sensor network. In *Electronics Computer Technology (ICECT), 2012 4th International Conference on*, volume 3, pages 50 –54, april 2012. ISBN 978-1-4673-1849-5.

[105] Mobility for Long Term Water Quality Monitoring. http://www.mobesens.eu/.

[106] J. Rousselot, Ph. Dallemagne, and Decotignie J-D. Deployments of wireless sensor networks performed by csem. In *COGNITIVE SYSTEMS WITH INTERACTIVE SENSORS*, 2009.

[107] A. El-Hoiydi and J.D. Decotignie. Wisemac: An ultra low power mac protocol for multi-hop wireless sensor networks. In *ALGOSENSORS*, pages 18–31, 2004.

[108] A. El-Hoiydi, C. Arm, R. Caseiro, S. Cserveny, J.-D. Decotignie, C. Enz, F. Giroud, S. Gyger, E. Leroux, T. Melly, V. Peiris, F. Pengg, P.-D. Pfister, N. Raemy, A. Ribordy, D. Ruffieux, and P. Volet. The ultra low-powerwisenet system. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1 –6, march 2006.

[109] Tahoe-LAFS. http://tahoe-lafs.org/trac/tahoe-lafs.

[110] É. Renault and D. Zeghlache. The metalist model: A simple and extensible information model for the future internet. In *EUNICE*, pages 88–97, 2009.

[111] ICEfaces, Open Source Ajax, J2EE Ajax, JSF Java Framework. http://www.icefaces.org/main/home.

[112] ISFET: Ion-Sensitive Field-Effect Transistor. http://en.wikipedia.org/wiki/ISFET.

[113] G.H. Zhang, C.C.Y. Poon, and Y.T. Zhang. A review on body area networks security for healthcare. *CN*, 2011:21:1–21:8, January 2011.

[114] B.P.L. Lo and G.-Z. Yang. Key Technical Challenges and Current Implementations of Body Sensor Networks. In *International Workshop on Wearable and Implantable Body Sensor Networks*, April 2005.

[115] C.C. Tan, HV Wang, S. Zhong, and Q. Li. Ibe-lite: A lightweight identity-based cryptography for body sensor networks. *Information Technology in Biomedicine, IEEE Transactions on*, 13(6):926 –932, nov. 2009.

[116] W. Drira, É. Renault, and D. Zeghlache. A hybrid authentication and key establishment scheme for wban. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 78 –83, june 2012.

[117] Y. Hovakeemian, K. Naik, and A. Nayak. A survey on dependability in body area networks. In *Proc. 5th Int Medical Information & Communication Technology (ISMICT) Symp*, pages 10–14, 2011.

[118] S. Josefsson. PKCS #5: Password-Based Key Derivation Function 2 (PBKDF2) Test Vectors. RFC 6070 (Informational), January 2011.

[119] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, and DE. Culler. Spins: security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, September 2002.

[120] K.G. Paterson and G. Price. A comparison between traditional public key infrastructures and identity-based cryptography. *Information Security Technical Report*, 8:57–72, 2003.

[121] A. Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[122] F. Hess. Efficient identity based signature schemes based on pairings. In Kaisa Nyberg and Howard Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer Berlin / Heidelberg, 2003.

[123] A. Joux. A one round protocol for tripartite diffie-hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[124] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. of Computing*, 32(3):586–615, 2003. extended abstract in Crypto'01.

[125] N. Gura, A. Patel, A. Wander, H. Eberle, and S.C. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *CHES*, pages 119–132, 2004.

[126] R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptography: A survey.

[127] M. Barua, M.S. Alam, X. Liang, and X. Shen. Secure and quality of service assurance scheduling scheme for wban with application to ehealth. In *Wireless Communications and Networking Conference (WCNC), 2011 IEEE*, pages 1102 –1106, march 2011.

[128] J. Liu and K.S. Kwak. Hybrid security mechanisms for wireless body area networks. In *Ubiquitous and Future Networks (ICUFN), 2010 Second International Conference on*, pages 98 –103, june 2010.

[129] A. Boudguiga and M. Laurent. An id-based authentication scheme for the ieee 802.11s mesh network. In *WiMob*, pages 256–263, 2010.

[130] G. YANG, C. RONG, C. VEIGNER, J. WANG, and H. CHENG. Identity-based key agreement and encryption for wireless sensor networks. *The Journal of China Universities of Posts and Telecommunications*, 13(4):54 – 60, 2006.

[131] C12G Labs S.L. Designing and Installing your Cloud Infrastructure. Technical report.

[132] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[133] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. *SIGCOMM Comput. Commun. Rev.*, 39(4):135–146, August 2009.

[134] S. Alshehri, S.P. Radziszowski, and R.K. Raj. Secure access for healthcare data in the cloud using ciphertext-policy attribute-based encryption. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 143 – 146, april 2012.

[135] L. Ibraimi, M. Asim, and M. Petkovic and. Secure management of personal health records by applying attribute-based encryption. In *Wearable Micro and Nano Technologies for Personalized Health (pHealth), 2009 6th International Workshop on*, pages 71 –74, june 2009.

[136] D. Guinard, M. Fischer, and V. Trifa. Sharing using social networks in a composable web of things. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 702 –707, 29 2010-april 2 2010.